Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Simon Ivanšek

Obremenitveno testiranje aplikacij v javnih oblakih

magistrsko delo

študijski program druge stopnje
računalništvo in informatika

Mentor: doc. dr. Mojca Ciglarič

Ljubljana, 2015

Izjava o avtorstvu magistrskega dela

Spodaj podpisani Simon Ivanšek sem avtor magistrskega dela z naslovom:

Obremenitveno testiranje aplikacij v javnih oblakih

S svojim podpisom zagotavljam, da:

- sem magistrsko delo izdelal samostojno pod mentorstvom doc. dr. Mojce Ciglarič,
- so elektronska oblika magistrskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko magistrskega dela,
- soglašam z javno objavo elektronske oblike magistrskega dela v zbirki "Dela FRI".

V Ljubljani, 24. september 2015 Podpis avtorja:

Zahvaljujem se svoji mentorci doc. dr. Mojci Ciglarič za usmeritve in konstruktivne kritike pri izdelavi magistrske naloge. Zahvalil bi se tudi punci in staršem za spodbudo in podporo skozi celoten študij.

Kazalo

Povzetek

V magistrski nalogi se ukvarjamo s problemom testiranja skalabilnosti in stroškov povezanih s povečanim obiskom, s katerim se danes srečuje vse več podjetij. Pri testiranju se osredotočimo na javna oblaka Amazon Web Services in Google Compute Platform. K reševanju problema pristopimo z obremenitvenim testiranjem tako, da razvijemo vzorčno aplikacijo in orodja, s katerimi jo obremenitveno testiramo. Vzorčno aplikacijo in orodja razvijemo po standardu TPC-W, ki ga uporabimo tudi za izvajanje obremenitvenega testa. Po zgledu metrik iz standarda TPC-W definiramo dve novi metriki, s katerima merimo in vrednotimo skalabilnost in stroške, povezane s povečanim obiskom. Meritve izvedemo najprej z ročnim skaliranjem, nato še z vkloplje-nim avtomatskim skaliranjem. Rezultate meritev ovrednotimo z definiranimi metrikami, izdelamo stroškovni model za omenjena oblaka in ju med seboj primerjamo.

Abstract

In this master thesis we are solving the problem of scalability testing and estimating the costs associated with increased load on web application, with which many companies are facing nowadays. We focus on testing of Amazon Web Services and Google Cloud Platform public clouds. Our approach to solving the problem is with load testing, that's why we develop sample web application and tools to execute the load tests. For developing the web application and tools we used TPC-W standard. We also execute the load test according to TPC-W standard. Based on TPC-W metrics, we develop two new metrics that we use for evaluating load tests. First we perform the measurements with manual scaling and then with automated scaling. After that we evaluate the measurements with newly defined metrics, make the cost model for each cloud and compare them.

Poglavje 1

Uvod

Z razvojem naprednih mobilnih tehnologij se je razvilo veliko podjetij, ki ponujajo različne storitve, ki nadomeščajo ali olajšujejo vsakodnevna opravila. Z vse večjo dostopnostjo do spleta se je tudi število uporabnikov teh storitev ekstremno povečalo. Uporabniki se v splet povezujejo preko različnih medijev (osebni računalnik, tablični računalnik, mobilni telefon, pametna ura) in zahtevajo neprekinjeno ter nemoteno delovanje storitev. V veliko pomoč podjetjem, da podprejo povečano število uporabnikov njihovih storitev in kljub temu ponujajo kvalitetne storitve z neprekinjenim delovanjem, je računalništvo v oblaku.

1.1 Opis problema in motivacija

V zadnjih letih je računalništvo v oblaku postalo eno glavnih področij raziskovanja v industriji IT. Uvaja revolucionaren pristop oskrbovanja in upravljanja z računalniškimi viri. Podjetjem ni potrebno več skrbeti za nakup, vzdrževanje in upravljanje dragih strežnikov. Računalništvo v oblaku z modelom plačevanja po porabi in oskrbovanja z računalniškimi viri na zahtevo ponuja storitve, ki zmanjšujejo stroške upravljanja in nameščanja aplikacij. Aplikacijam v teoriji z neomejenimi računalniškimi viri in avtomatskim dodajanjem ter odvzemanjem računalniških virov, ponuja neskončno skalabilnost.

Skaliranje računalniških virov aplikacijam omogoča podporo velikemu številu sočasnih uporabnikov, ki v današnjem času zahtevajo hitro in neprekinjeno delovanje.

Podjetja se za izbiro oblaka pogosto odločajo na podlagi cene, saj večina oblakov ponuja podobne in podobno zmogljive storitve. Ponudniki računalništva v oblaku na svojih spletnih straneh pogosto ponujajo kalkulatorje [1], ki omogočajo primerjanje cen s konkurenčnimi ponudniki. Problem takih kalkulatorjev je, da ponujajo cenovno primerjavo za fiksne namestitvene konfiguracije, ne pa tudi cenovne primerjave skalabilnosti oz. namestitvenih konfiguracij ob povečani obremenitvi.

V magistrski nalogi se ukvarjamo s problemom testiranja skalabilnosti spletnih aplikacij in stroškov, povezanih s povečanim obiskom. S tem namenom se postavimo v vlogo podjetja, ki želi preko spleta prodajati knjige, za izvajanje spletne knjigarne pa uporablja javni oblak. Kot podjetje nas zanima, koliko uporabnikov bomo lahko podprli v primeru, da se obisk spletne knjigarne nenadoma poveča, npr. med božično-novoletnimi prazniki, in koliko bomo za to plačali. Ukvarjamo se z vprašanjem, kako objektivno ovrednotiti in izmeriti lastnosti različnih javnih ponudnikov ter jih med seboj cenovno primerjati. S tem namenom definiramo metrike in razvijemo orodja za merjenje in vrednotenje obremenitvenih testov z definiranimi metrikami.

1.2 Pregled sorodnih del

V literaturi obstaja zelo malo del, ki bi se ukvarjala z analiziranjem skalabilnosti in stroškov, povezanih s povečano obremenitvijo. Kljub pomanjkanju literature obstaja nekaj raziskav, kjer avtorji rešujejo podobne probleme. V nadaljevanju razdelka opišemo nekaj pomembnejših del in njihovih pristopov, ki so nam služila kot izhodišče za reševanje omenjenega problema.

Xu et al. [2] pristopijo k obremenitvenemu testiranju z analizo dnevnika spletnih dostopov (angl. *web access log*). Avtorji članka nadgradijo model Customer Behavior Model Graph (CBMG), ki ima to slabost, da med eval-

vacijo modela lahko pride do napak zaradi topoloških razmerij med zapisi v dnevniku spletnih dostopov. Model CBMG avtorji nadgradijo tako, da zgradijo verjetnostni matriki, eno za verjetnost prehoda med statusi, drugo pa za čase razmišljanja uporabnika. Model se nato uporabi za ustvarjanje 80 virtualnih uporabnikov in simulacijo, ki se izvaja 10 minut. Rezultat simulacije je graf v notaciji končnih avtomatov, kjer vozlišča predstavljajo statuse, na prehodih pa je zapisan par (verjetnost prehoda, čas razmišljanja uporabnika).

Gao et al. [3] celovito pregledajo področje testiranja v oblaku, tako da govorijo o problemih in izzivih testiranja v oblaku ter jih razdelijo v 4 glavne skupine: vzpostavitev testnega okolja na zahtevo, testiranje skalabilnosti in performančnih lastnosti, testiranje in merjenje varnosti v oblaku in integracijsko testiranje v oblaku. Avtorji se dotikajo tudi glavnih potreb testiranja v oblaku: ustrezni testni modeli in kriteriji, definicija procesov in standardov za merila kakovosti testiranja v oblaku, inovativne testne metode in rešitve ter inovativne dinamične testne platforme in orodja. Avtorji za konec tudi na kratko primerjajo zmogljivost različnih že obstoječih orodij za testiranje aplikacij v oblaku: PushtoTest [4], Cloud Testing [5], SOASTA [6] in iTKO [7].

Zhang et al. [8] predlagajo merilo (angl. *benchmark*) za spletne trgovine Bench4Q, ki temelji na merilnem standardu TPC-W [9]. Merilo Bench4Q uporabimo za uravnavanje nastavitev strežnika, na katerem se izvaja spletna trgovina. Standard TPC-W avtorji članka nadgradijo z metriko, ki temelji na sejah namesto na zahtevkih HTTP. Zaprt način simuliranja, kjer vsak uporabnik izvede eno operacijo znotraj seje, nadgradijo z odprtim, kjer en uporabnik izvede zaporedje operacij znotraj seje.

Transaction Performance Council, ki ga sestavljajo podjetja, kot so Intel, IBM, Google in ostala, so leta 2000 definirala standard TPC-W [9]. Standard TPC-W je merilo za spletne trgovine in določa obremenitev, ki simulira uporabnike, ki brskajo in nakupujejo produkte na spletni strani. Standard definira 14 operacij in za vsako od njih odzivni čas, ki ga mora dobiti si-

mulator brskalnika od sistema, ki ga testiramo. Standard TPC-W natančno določa implementacijo vsake od 14 operacij in arhitekturo celotnega sistema. Standard TPC-W definira tudi dve metriki: WIPS (Web Interactions Per Second) in ceno na WIPS. WIPS je število interakcij na sekundo, ki jih lahko sistem, ki ga testiramo še zdrži v okviru določenih odzivnih časov. Cena na WIPS je cena celotnega sistema, ki ga testiramo, deljeno z WIPS.

Standard TPC-W je od leta 2005 uradno označen za zastarelega, vendar ga v literaturi še vedno veliko uporabljajo kot osnovo za raziskovanje problemov, povezanih z obremenitvenim testiranjem.

Pri izdelavi magistrske naloge smo za osnovo uporabili standard TPC-W, ker je po vsebini še najbližje problemu, ki ga poskušamo rešiti. Ideje za definicijo metrik smo v glavnem črpali iz del [10] in [11].

Poglavje 2

Računalništvo v oblaku

Računalništvo v oblaku je način nudenja računalniških virov (aplikacij, omrežij, trdih diskov itd.) preko spleta na zahtevo. Uporabnikom omogoča hitro oskrbovanje in sproščanje potrebnih računalniških virov z modelom plačevanja "plačaj po porabi"(angl. *pay-as-you-go*). Koncept računalništva v oblaku temelji na mrežnem računalništvu (angl. *grid computing*), porazdeljenem računanju (angl. *distributed computing*) in računanju v gručah (angl. *cluster computing*). Obstajata dva glavna namestitvena modela računalništva v oblaku, in sicer **javni** in **privatni**. Podjetja pogosto uporabljajo kombinacijo obeh in takrat govorimo o **hibridnem** oblaku.

Računalništvo v oblaku spreminja tradicionalni silosni model, kjer so procesi in podatki organizirani kot posamezna enota neodvisna od ostalih, v model storitev, kjer so procesi in podatki povezani med seboj preko neodvisnih enot, katere funkcionalnosti so dostopne preko storitve. Da bi razumeli temelje računalništva v oblaku, je potrebno razumeti nivoje, ki ga sestavljajo:

- IaaS - Infrastructure as a Service (slo. Infrastruktura kot storitev),
- PaaS - Platform as a Service (slo. Platforma kot storitev),
- SaaS - Software as a Service (slo. Programska oprema kot storitev).

IaaS ponuja dostop do strojne opreme v virtualizarnem okolju kot storitev, dostopno preko spleta. Uporabniku omogoča vertikalno (dodajanje računalniških virov) in horizontalno skaliranje (podvajanje računalniških virov) z zelo nizkimi stroški in brez skrbi za fizično delovanje strojne opreme, ki je prepuščena ponudniku storitev.

PaaS ponuja platformo, orodja in okolje, ki razvijalcem programske opreme pomagajo pri razvoju. Storitve PaaS se izvajajo v oblaku, uporabniku pa so dostopne preko spletnega brskalnika, vmesnika za ukazno vrstico ali programskega vmesnika. Primeri storitev PaaS so:

- storitev za upravljanje s programsko opremo (OpenShift),
- storitev za upravljanje s podatkovnimi bazami,
- storitev za podatkovno shranjevanje.

SaaS opisuje vsako storitev, pri kateri lahko uporabniki dostopajo do programske opreme preko spleta. Programska oprema se izvaja in gostuje v oblaku ter za svoje delovanje uporablja storitve IaaS nivoja, lahko tudi storitve PaaS nivoja. Primeri storitev SaaS so spletne aplikacije Gmail, Salesforce in ZenDesk.

Poglavje 3

Javni oblak

V tem poglavju predstavimo dva ponudnika javnega oblaka, Amazon Web Services (AWS) in Google Compute Platform (GCP), na katera smo namestili spletno trgovino, jo obremenitveno testirali in ovrednotili rezultate z definiranimi metrikami. Predstavimo principe delovanja storitev posameznega oblaka, ki smo jih uporabili za nameščanje in testiranje spletne trgovine v izbran oblak.

3.1 Amazon Web Services

Amazon Web Services (AWS) je zbirka spletnih storitev, ki jih ponuja podjetje Amazon in omogoča razvijalcem programske opreme njihovo uporabo. Zbirka vključuje številne infrastrukturne storitve, ki simulirajo ali nadomeščajo tradicionalno fizično infrastrukturo, ki je potrebna za izvajanje spletnih aplikacij. Infrastrukturne storitve vključujejo storitve za:

- shranjevanje podatkov,
- računsko moč,
- sporočilne sisteme,
- plačilne sisteme,

- podatkovne baze,
- analitiko,
- omrežje,
- monitoring,
- mobilno poslovanje,
- poslovne aplikacije.

Infrastrukturne storitve lahko uporablja vsak z Amazonovim uporabniškim računom in bančno kartico. Uporabnik teh storitev plača samo toliko, kolikor porabi, ne glede na to ali samo eksperimentira s storitvami ali jih uporablja za podporo tisočim uporabnikom, ki uporabljajo njegovo spletno aplikacijo.

Storitve oblaka AWS ponujajo dobro alternativo razvoju aplikacij na lastni fizični strojni opremi, katere nakup in vzdrževanje zahteva veliko truda in stroškov. Storitve na oblaku AWS nudijo skalabilne, zanesljive in cenovno učinkovite storitve, ki jih razvijalci programske opreme potrebujejo za izvajanje in razvoj svojih aplikacij. Storitve oblaka AWS močno zmanjšajo začetno investicijo, ki je potrebna za razvoj in zagon spletne aplikacije, pri tem pa zagotavljajo, da bo aplikacija zmožna ne samo preživeti nenadno pozornost, ampak tudi zmožna rasti in dobre odzivnosti. S prepuščanjem obvladovanja infrastrukture Amazonu, se lahko razvijalci osredotočijo na razvoj aplikacij in poslovno rast podjetja [12].

Vse storitve oblaka AWS lahko uporabljamo preko spletnega grafičnega vmesnika, vmesnika za ukazno vrstico (angl. *command line interface - CLI*)in programskega vmesnika (API).

3.1.1 Elastic Compute Cloud

Elastic Compute Cloud (EC2) je ena izmed spletnih storitev oblaka AWS, ki omogoča oskrbovanje z virtualnimi stroji različnih zmogljivosti in

različnimi operacijskimi sistemi. Storitev je sestavljena iz podstoritev, ki so logično razdeljene v naslednje skupine:

- instance (angl. *instances*),
- slike (angl. *images*),
- elastična blokovna shramba (angl. *elastic block storage*),
- omrežje in varnost (angl. *network & storage*),
- izenačevanje obremenitve (angl. *load balancing*),
- avtomatsko skaliranje (angl. *auto scaling*).

Skupina: instance

V skupini *instance* lahko ustvarjamo, pregledujemo, spreminjamo in brišemo virtualne stroje. Storitev EC2 razdeli virtualne stroje v različne skupine, glede na namen uporabe, znotraj skupin pa še glede na zmogljivost. Glavne skupine so:

Instance tipa T2 imajo t. i. eksplozivno zmogljivost (angl. *burstable performance*), ki ponuja osnovni nivo CPU zmogljivosti z možnostjo povečanja nad osnovnim nivojem. Osnovni nivo CPU zmogljivosti in možnost povečane zmogljivosti se nadzoruje s CPU točkami. Vsaka instanca tipa T2 prejme določeno število CPU točk na uro, odvisno od tipa instance. Instance tipa T2 pridobivajo CPU točke v mirovanju in jih koristijo, ko so aktivne. Instance tipa T2 so dobra izbira za obremenitve, ki pogosto ne uporabljajo celotnega CPU. Namenjene so testiranju in razvojnemu okolju ter neprimerne za produkcijsko uporabo.

Instance tipa M3 in M4 so primerne za okolja, kjer potrebujemo ravnovesje med računskimi, spominiskimi in omrežnimi viri. Običajno je izbira instanc tipa M4 in M3 dobra izbira za večino aplikacij.

Instance tipa C3 in C4 so instance, ki so optimizirane za računsko moč. Imajo najbolj zmogljive procesorje in nanižjo ceno glede na računsko zmogljivost.

Instance tipa R3 so instance, ki so optimzirane za spominsko intezivne aplikacije in imajo najnižjo ceno glede na gigabajt RAM-a med vsemi instancami EC2.

Instance tipa G2 so instance optimizirane za računsko intezivne aplikacije, kjer se računski del izvaja na grafični kartici.

Instance tipa I2 ponujajo SSD podprto shranjevanje podatkov za zelo veliko vhodno/izhodno zmogljivost po ugodni ceni.

Instance D2 ponujajo vse do 48 TB podatkovne shrambe, z visoko prepustnostjo in majhno ceno glede na zmogljivost prepustnosti na podatkovni disk.

Tabela vseh razpoložljivih instanc in njihovih zmogljivosti je na voljo na naslovu [13].

Skupina: slike

Skupina *slike* nam omogoča izdelavo slik virtualnih strojev, ki jih lahko uporabimo kasneje, ali pa delimo z ostalimi uporabniki storitev EC2. Slika virtualnega stroja se v storitvi EC2 imenuje Amazon Machine Image (AMI). Slika virtualnega stroja vsebuje celoten operacijski sistem in vso programsko opremo, ki smo jo namestili na virtualni stroj. Izdelava slike virtualnega stroja nam omogoča, da se izognemo vsakokratni namestitvi in nastavitvi programske opreme na virtualni stroj, ko jo potrebujemo.

Skupina: elastična blokovna shramba

Skupina *elastična blokovna shramba* predstavlja skupino, v kateri se nahajajo logični podatkovni diski, ki so pripeti virtualnim strojem. V skupini najdemo tudi posnetke (angl. *snapshots*) logičnih podatkovnih diskov, ki jih lahko ustvarimo.

Skupina: omrežje in varnost

Skupina **omrežje in varnost** vsebuje **varnostne skupine**, **elastične naslove IP**, **omrežne vmesnike** in **varnostne ključe**.

Vsak virtualni stroj mora imeti vsaj en logični omrežni vmesnik, preko katerega je možna povezava na virtualni stroj. Omrežnemu vmesniku se avtomatsko dodeli interni naslov IP, ki je unikaten za celoten podatkovni center, v katerem se izvaja naš virtualni stroj. Če želimo, da je virtualni stroj dostopen tudi od zunaj, mu dodelimo **elastični naslov IP**. **Elastični naslov IP** je javni naslov IP za dostop do virtualnega stroja.

Po privzetem je javni dostop do virtualnega stroja onemogočen na vseh vratih, zato moramo na primer za dostop do spletnega strežnika, ki se običajno izvaja na vratih 80, eksplicitno omogočiti dostop do teh vrat. To naredimo tako, da ustvarimo **varnostno skupino**, v kateri povemo kdo (naslov IP) in preko katerih vrat ima dostop do virtualnega stroja.

Za SSH dostop do virtualnega stroja moramo poleg **varnostne skupine** ustvariti še **varnostni ključ**, ki ga uporabimo pri povezovanju na virtualni stroj.

Skupina: izenačevanje obremenitve

Storitev **izenačevanja obremenitve** (angl. *load balancing*) omogoča razporejanje spletnega prometa med več virtualnih strojev. Izenačevalcu obremenitve povemo, po katerem protokolu (HTTP, TCP, HTTPS, SSL) in na katera vrata naj razporeja promet. Povemo mu tudi, na koliko časa (v sekundah) naj preverja razpoložljivost virtualnega stroja, kdaj naj ga odstrani

iz množice virtualnih strojev v primeru nerazpoložljivosti itd. Strategija, po kateri izenačevalec obremenitve razporeja promet med virtualnimi stroji, je Round-robin.

Skupina: avtomatsko skaliranje

V skupini **avtomatsko skaliranje** imamo na voljo storitve za nastavljanje in urejanje politik za avtomatsko skaliranje. Avtomatsko skaliranje je sestavljeno iz dveh delov. Prvi del sestavljajo t. i. *zagonske nastavitve* (angl. *launch configuration*), ki povedo storitvi avtomatskega skaliranja, kako in kakšen virtualni stroj naj ustvari v primeru povečane obremenitve. Drugi del predstavljajo t. i. *skupine za avtomatsko skaliranje* (angl. *autoscaling groups*), ki določajo, katero *zagonsko nastavitev* naj uporabijo, politiko skaliranja gor in dol, minimalno število virtualnih strojev ob zmanjšani obremenitvi in maksimalno število virtualnih strojev ob povečani obremenitvi.

Politika skaliranja temelji na storitvi CloudWatch, ki meri in spremlja različne lastnosti virtualnega stroja:

- izkoriščenost CPU,
- stanje CPU kreditov,
- izhodni omrežni promet,
- vhodni omrežni promet,
- število zapisanih podatkov na disk,
- število prebranih podatkov z disk.

Pri politiki skaliranja definiramo tudi t. i. čas ohlajanja (angl. *cooldown time*). To je čas, ki je potreben, da na novo dodani viri v celoti prevzamejo svojo funkcijo. V čas ohlajanja je vključen čas zagona virtualnega stroja, čas zagona operacijskega sistema in čas izvajanja programov, ki se zaženejo ob zagonu virtualnega stroja. Privzeto je čas nastavljen na 5 minut. Če je

v tem času sistem videti preobremenjen, nima smisla dodati še novih dodatnih virov, saj še prejšnji novo dodani viri niso v celoti funkcionalni. Šele, ko preteče čas ohlajanja in novo dodani viri v celoti delujejo, je smiselno nadaljevati s skaliranjem, torej z dodajanjem novih virov, če se to izkaže za potrebno.

Da je avtomatsko skaliranje nevidno za končnega uporabnika storitev, uporablja tudi storitev **izenačevanja obremenitve**. Storitev izenačevanja obremenitve ponuja enoten URL naslov za dostop do množice virtualnih strojev. Storitev avtomatskega skaliranja z dodajanjem in odstranjevanjem virtualnih strojev prilagaja velikost množice virtualnih strojev trenutni obremenitvi.

3.1.2 Storitev za relacijske podatkovne baze

Storitev za relacijske podatkovne baze (angl. *Relational Database Service - RDS*) je storitev oblaka AWS za porazdeljene podatkovne baze, ki omogoča oskrbovanje, upravljanje in skaliranje različnih relacijskih podatkovnih baz: MySQL, Postgres, Oracle in Microsoft SQL Server. Kompleksni administratorski procesi nameščanja popravkov, varnostnega kopiranja podatkovne baze in obnovitve podatkovne baze v določeno časovno točko, so avtomatski in enostavnejši za uporabnika [14].

Lastnosti storitve za relacijske podatkovne baze

Več razpoložljivostnih območij Podatkovno bazo lahko namestimo v več razpoložljivostnih območij. Ta oblika namestitve je primerna za produkcijska okolja. Namen te oblike namestitve je povečati razpoložljivost in obstojnost podatkov za podatkovno bazo. Ko zaženemo podatkovno bazo v več razpoložljivostnih območjih, storitev RDS avtomatsko ustvari in upravlja sinhrono repliko podatkovne baze v različnih razpoložljivostnih območjih (neodvisna infrastruktura na fizično ločeni lokaciji). V primeru omenjene namestitve imamo v primeru izpada celotnega podatkovnega centra še vedno na voljo delujočo podatkovno bazo v drugem podatkovnem centru.

Možnost zagona instance virtualnega stroja v več razpoložljivostnih območjih je opcijska, z njo pa so povezani tudi dodatni stroški. Ko uporabnik preko spletnega uporabniškega vmesnika ustvarja instanco RDS, ga čarovnik vpraša ali želi zagnati instanco v več razpoložljivostnih območjih [14].

Replike za branje Storitev RDS omogoča enostavno izdelavo replik za branje za podatkovno bazo MySQL. Replike za branje so instance podatkovne baze MySQL, ki vsebujejo enako kopijo podatkov kot glavna instanca in so namenjene samo bralnim operacijam. Gonilnik za podatkovno bazo na strani aplikacije, ki se povezuje na podatkovno bazo poskrbi, da se pisalne operacije izvedejo na glavni instanci, bralne operacije pa se po metodi Round-robin razporedijo tudi med replike za branje.

Tipi instanc Instance storitve RDS so razdeljene v 7 različnih zmogljivostnih skupin glede na tip uporabe podobno kot instance EC2. Instance so omejene z računsko močjo, pomnilnikom in zmogljivostjo omrežja. Zelo pomembna omejitev, ki je Amazon ne oglašuje na svojih straneh, je nastavitev podatkovne baze MySQL **max_connections**, ki določa maksimalno število sočasnih povezav na podatkovno bazo. Te nastavitve se ne da spreminjati, njena vrednost pa je odvisna od tipa instance. Celotna tabela tipov instanc in njihovih zmogljivosti je na voljo na naslovu [15].

3.1.3 Simple Storage Service

Storitev Amazon Simple Storage Service (S3) ponuja porazdeljeno, skalabilno in varno spletno shranjevanje kakršnih koli podatkov. Storitev ne omejuje števila in velikosti podatkov, časa shranjevanja podatkov ali pasovne širine prenosa podatkov, saj uporabnik plača samo za to, kar porabi [12].

Podatki na S3 so organizirani v koše (angl. *buckets*), ki jim lahko nastavljamo različne pravice dostopa, pregledujemo dnevnik dostopov za koše in njihove objekte ter izberemo regijo, v kateri se nahaja koš, da zmanjšamo latenco in stroške [16].

3.2 Google Cloud Platform

Google Cloud Platform (GCP) je oblak, ki ga ponuja podjetje Google na isti infrastrukturi, kot jo uporablja za izvajanje svojih produktov, kot so Google iskanje, Gmail in YouTube. Oblak ponuja različne storitve za gradnjo, testiranje in nameščanje aplikacij na Googlovo visoko-skalabilno in zanesljivo infrastrukturo. Uporabniki lahko izbiramo med storitvami za shranjevanje podatkov, storitvami za računsko moč in aplikacijskimi storitvami za spletne, mobilne in poslovne rešitve [17].

Google Cloud Platform je sestavljen iz množice storitev, kjer vsaka nudi spletni vmesnik, vmesnik za ukazno vrstico in programski vmesnik REST. Storitve so razdeljene v naslednje skupine:

- Compute:
  - Compute Engine,
  - App Engine,
  - Container Engine,
- Storage:
  - Storage,
  - Bigtable,
  - Datastore,
  - Cloud SQL,
- Networking:
  - Load balancing,
  - Interconnect,
  - DNS,
- Big Data:

  - BigQuery,
  - Dataflow,
  - Pub/Sub,

- Services:
  - Translate API,
  - Prediction API,
  - Endpoints,

- Management:
  - Deployment Manager,
  - Logging,
  - Monitoring.

Skupine in njihove storitve, ki smo jih uporabili za izdelavo magistrske naloge, so podrobneje opisane v nadaljevanju.

3.2.1 Google Compute Engine

Google Compute Engine (GCE) sestavlja množica storitev za oskrbovanje z virtualnimi stroji, omrežjem, avtomatskim skaliranjem in podatkovnimi diski. Vsaka storitev je dostopna preko spletnega vmesnika, vmesnika z ukazno vrstico in programskega vmesnika REST. Uporabniki lahko ustvarijo virtualne stroje z različnimi nastavitvami in različnimi operacijskimi sistemi. Podatkovni disk virtualnega stroja je shranjen na trajnem blokovnem pomnilniku, ki je podvojen za redundanco in ga lahko uporabimo tudi po tem, ko se virtualni stroj ne izvaja več. Omrežni dostop lahko nastavimo, tako da dovolimo dostop samo med virtualnimi stroji, iz spleta ali samo iz privatnega omrežja [18]. Storitev je razdeljena v skupine:

- instance virtualnih strojev (angl. *VM instances*),

- skupine instanc (angl. *Instance Groups*),
- predloge instance (angl. *Instance Templates*),
- podatkovni diski (angl. *Disks*),
- posnetki (angl. *Snapshots*),
- slike (angl. *Images*),
- meta podatki (angl. *Metadata*),
- izenačevanje obremenitve (angl. *Load balancer*),
- območja (angl. *Zones*),
- operacije (angl. *Operations*),
- kvote (angl. *Quotas*),
- nastavitve (angl. *Settings*).

Skupina: Instance virtualnih strojev

V skupini **instance virtualnih strojev** lahko ustvarjamo, pregledujemo, brišemo in spreminjamo virtualne stroje. Virtualni stroji so lahko različnega tipa, ki virtualnemu stroju določa njegove zmogljivosti. Storitev GCE razdeli tipe virtualnih strojev na 4 glavne skupine:

- stroji z deljenim CPU (oznaki f1-micro in g1-small),
- standardni stroji (oznaka n1-standard),
- stroji z visoko-zmogljivim CPU (oznaka n1-highcpu),
- stroji z visoko-zmogljivim RAM (oznaka n1-highmem).

Stroji z deljenim CPU so primerni za aplikacije, ki ne zahtevajo veliko virov. Virtualni stroji z deljenim jedrom CPU so cenovno bolj učinkoviti za izvajanje majhnih, nezahtevnih aplikacij kot pa standardni stroji, stroji z visoko-zmogljivim CPU ali stroji z visoko-zmogljivim RAM-om. V tej skupini sta dva tipa virtualnih strojev:

- f1-micro (1 vCPU, 0.6 GB RAM, deljen CPU),
- g1-small (1 vCPU, 1.70 GB RAM, 1.38 GCEUs. [1]

Virtualni stroj tipa *f1-micro* ima t. i. *eksplozivno zmogljivost* podobno kot virtualni stroj tipa *t2* pri storitvi EC2. Gre za koncept, kjer instanca tega tipa v primeru povečane porabe CPU za nekaj časa dobi povečano zmogljivost CPU preko osnovne.

Standardni stroji so primerni za produkcijsko uporabo, saj ponujajo najboljše razmerje računalniških virov po najboljši ceni.

Stroji z visoko-zmogljivim CPU so idealni za naloge, ki zahtevajo več virtualnega CPU relativno glede na RAM. Virtualni stroji tega tipa imajo en virtualni CPU na vsakih 0.90 GB RAM-a.

Stroji z visoko-zmogljivim pomnilnikom so idealni za naloge, ki zahtevajo več RAM-a relativno glede na CPU. Virtualni stroji tega tipa imajo 6.5 GB RAM-a na en virtualni CPU.

Skupina: Skupine virtualnih strojev

Skupine virtualnih strojev [2] nam omogoča kolektivno upravljanje skupin virtualnih strojev. Skupine virtualnih strojev moramo uporabljati, če želimo uporabljati storitev za avtomatsko skaliranje in storitev HTTP izenačevalca obremenitve.

[1] GCEU (Google Compute Engine Unit) je enota CPU kapacitete, kjer 2.75 GCEU enote ustreza enemu virtualnemu CPU.

[2] Storitev je v beta razvojni fazi. To pomeni, da SLA za to storitev ne velja.

Skupine virtualnih strojev imajo omejitev 1 000 000 poizvedb/dan in 20 poizvedb/sekundo.

Skupina: Slike

Skupina **slike** vsebuje slike virtualnih strojev, ki vsebuje zagonski nalagalnik (angl. *boot loader*), operacijski sistem in glavni datotečni sistem, ki je potreben za zagon virtualnega stroja. Vsakič, ko ustvarjamo virtualni stroj ali trajni podatkovni disk, moramo določiti sliko virtualnega stroja. Izdelava slike virtualnega stroja nam omogoča, da ne potrebujemo ob vsakem zagonu novega virtualnega stroja na novo namestiti in nastaviti programske opreme, ampak namesto tega pripravimo sliko virtualnega stroja z vnaprej nameščeno in nastavljeno programsko opremo.

Skupina: Izenačevanje obremenitve

Storitev Google Compute Engine ponuja upravljano izenačevanje obremenitve, tako da je vhodni omrežni promet porazdeljen med več virtualnih strojev. Izenačevanje obremenitve ima naslednje prednosti:

- skaliranje aplikacij,
- podpora veliki količini omrežnega prometa,
- zaznavanje "nezdravih"virtualnih strojev,
- izenačevanje obremenitve med regijami,
- usmerjanje prometa na najbližji virtualni stroj,
- podpora vsebinskemu usmerjanju (CSS datoteke, slike).

Oblak GCP ponuja dva tipa izenačevalca obremenitve. Prvi se imenuje **omrežni izenačevalec obremenitve**, drugi pa **izenačevalec obremenitve HTTP zahtev**.

Omrežni izenačevalec obremenitve se uporablja za izenačevanje prometa TCP in UDP ter uporablja **pravila posredovanja** (angl. *forwarding rule*) za določanje prometa, ki bo preusmerjen na **ciljni bazen**. Primer: **pravilo posredovanja** lahko določa, da samo promet TCP, namenjen naslovu IP 192.0.2.0.1 na vratih 80 preusmeri na **ciljni bazen**.

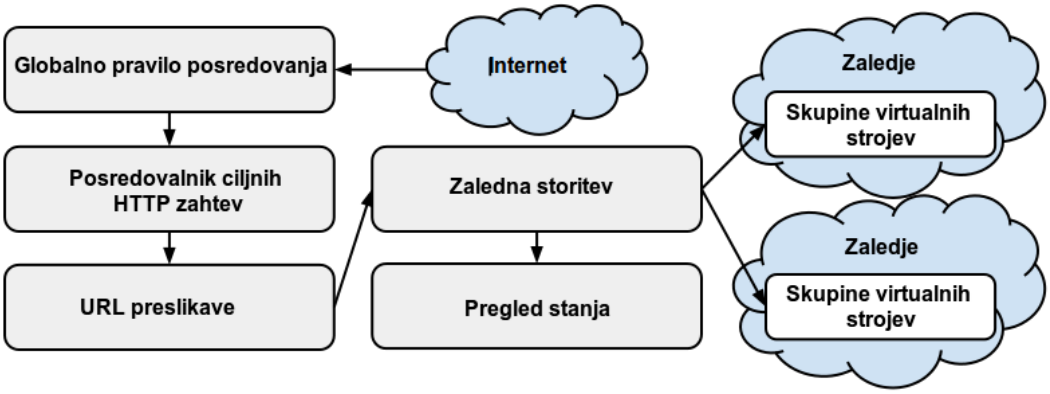
Ciljni bazen je množica virtualnih strojev, ki so pripravljeni sprejeti promet. Za določanje virtualnega stroja, ki bo sprejel promet, se uporablja storitev **pregleda stanja**.

Pregled stanja (angl. *health check*) zagotavlja storitvi izenačevanja obremenitve, razporejanje novih zahtevkov samo na virtualne stroje, ki se izvajajo in so pripravljeni na sprejem novih zahtevkov. Storitev **pregled stanja** preverja stanje virtualnih strojev, tako da pošilja zahtevo za pregled stanja na vsak virtualni stroj v določenih intervalih. Ko virtualni stroj preseže število neuspešnih zahtev za pregled stanja, se ga obravnava kot nezmožnega prejemanja novih zahtevkov in je odstranjen iz množice virtualnih strojev, ki lahko sprejemajo zahtevke. Tak virtualni stroj bo prejemal samo TCP pakete od že obstoječe vzpostavljenih povezav TCP, dokler te niso prekinjene ali ne ugasnemo virtualnega stroja. To dovoljuje virtualnim strojem počasno ustavljanje brez prekinjanja povezav TCP.

Zahteva za pregled stanja vrne vrednost HEALTHY ali UNHEALTHY. Da zahteva za pregled stanja vrne vrednost HEALTHY, mora zahteva vrniti HTTP odzivno kodo 200.

Pri ustvarjanju **pregleda stanja** določimo protokol (HTTP ali HTTPS), vrata (npr. 80) in pot (npr. /), ki želimo, da se preverja pri pregledu stanja virtualnega stroja. Ostale nastavitve za **pregled stanja** so še:

- interval preverjanja (privzeto: 5 sekund),
- časovna omejitev (angl. *timeout*) (privzeto: 5 sekund),
- število neuspešnih zahtev (privzeto: 2 zaporedni neuspešni zahtevi),
- število uspešnih zahtev (privzeto: 2 zaporedni uspešni zahtevi).



Slika 3.1: Delovanje izenačevalca obremenitve HTTP zahtev.

Izenačevalec obremenitve HTTP zahtev [3] je namenjen samo izenačevanju HTTP in HTTPS zahtev ter uporablja drugačne koncepte kot **omrežni izenačevalec obremenitve**. Izenačevalec obremenitve HTTP zahtev uporablja koncept **zaledne storitve** (angl. *backend service*), koncept **preslikav URL** (angl. *url map*) in **posredovalnik ciljnih HTTP zahtev** (angl. *target http proxy*).

Izenačevalec obremenitve HTTP zahtev deluje tako, da HTTP zahtevo sprejme *globalno pravilo posredovanja*, ki preusmeri zahtevek na *posredovalnik ciljnih HTTP zahtev*. Ta preveri zahtevek, če se ujema s kakšno *preslikavo URL*, ki zahtevek preusmeri na pravo *zaledno storitev*. *Zaledna storitev* zahtevek preusmeri na enega izmed razpoložljivih virtualnih strojev. Razpoložljivost virtualnih strojev preverja s pomočjo storitve *pregled stanja*. Slika 3.1 prikazuje opisano delovanje izenačevalca obremenitve HTTP zahtev.

Zaledna storitev definira eno ali več skupin instanc virtualnih strojev. Kapaciteto skupine določi glede na izkoriščenost CPU ali glede na število prihajajočih zahtevkov na sekundo. Vsaka zaledna storitev tudi določa enega ali več *pregledov stanja*, ki ga uporablja za preverjanje zdravih instanc virtualnih strojev. Instance virtualnih strojev se dodajajo v zaledno storitev z uporabo *skupine virtualnih strojev* (3.2.1).

[3] Storitev je v beta razvojni fazi. SLA za to storitev ne velja.

Preslikave URL definirajo pravila in ujemajoče vzorce za razporejanje zahtev na podlagi naslova URL na pravo *zaledno storitev*.

Posredovalnik ciljnih HTTP zahtev uporablja globalno *posredovalno pravilo* za razporejanje prihajajočih HTTP in HTTPS zahtevkov na *preslikave URL*.

Skupina: Območja

Google Cloud Platform viri gostujejo v več lokacijah po svetu. Lokacije so razdeljene na regije, regije pa na območja. Pri ustvarjanju virtualnega stroja lahko izberemo, v kateri regiji in območju želimo, da se virtualni stroj izvaja. Izbira regije in območja je pomembna, saj se lahko samo viri, ki so v istem območju in isti regiji uporabljajo med seboj. Primer: podatkovni diski se morajo nahajati v istem območju kot virtualni stroj, ki mu je pripet podatkovni disk. Podobno: statični naslov IP se mora nahajati v isti regiji, kot se nahaja virtualni stroj, da ga lahko virtualni stroj uporablja. Seznam regij in pripadajočih območij zajema tri lokacije v vzhodni Aziji, tri lokacije v zahodni Evropi in štiri centralne lokacije v ZDA.

Avtomatsko skaliranje

Storitev avtomatskega skaliranja je v oblaku GCP del storitve **upravljanih skupin instanc** (angl. *managed instance groups*). Upravljane skupine instanc so množice instanc, ustvarjenih iz predloge instanc (angl. *instance templates*). Storitev doda ali odstrani virtualne stroje iz **upravljanje skupine instanc**. Čeprav storitev GCE vsebuje upravljanje in neupravljanje skupine instanc, lahko avtomatsko skaliranje uporabljamo samo z upravljanimi skupinami instanc. Storitev avtomatskega skaliranja omogoča tri načine skaliranja:

- glede na povprečno izkoriščenost CPU virtualnih strojev,
- glede na metrike storitve Cloud Monitoring,

- glede na izkoriščenost ali število zahtev na sekundo izenačevalca obremenitve HTTP zahtev.

V magistrski nalogi smo uporabili skaliranje glede na povprečno izkoriščenost CPU virtualnih strojev, ki je tudi najpreprostejši način avtomatskega skaliranja. Omenjeni način skaliranja storitvi pove, da naj opazuje povprečno izkoriščenost CPU virtualnih strojev in jih na podlagi tega doda ali odstrani z namenom vzdrževanja določene izkoriščenosti CPU [18].

Pri ustvarjanju in določanju politike skaliranja lahko določimo t. i. čas ohlajanja (angl. *cooldown time*), ki storitvi avtomatskega skaliranja pove, koliko sekund naj počaka, predno začne zbirati podatke o izkoriščenosti CPU. Čas ohlajanja smo nastavili na isto vrednost kot na oblaku AWS, na 300 sekund.

Če je izkoriščenost CPU blizu 100 %, storitev avtomatskega skaliranja oceni, da je skupina instanc preobremenjena, zato storitev poveča število virtualnih strojev za vsaj 50 % ali minimalno za 4 instance. Storitev uporabi kriterij, ki je po vrednosti povečanja števila virtualnih strojev večji. Na ta način storitev avtomatskega skaliranja zagotavlja, da povprečna izkoriščenost CPU v skupini instanc nikoli ne preseže 100 %. V dokumentaciji na spletni strani [18] eksplicitno navajajo, da je to trenutno obnašanje in se lahko v prihodnosti spremeni.

Storitev **avtomatskega skaliranja navzdol** (angl. *downscale*) pogleda zadnjih 10 minut izkoriščenosti CPU virtualnega stroja, predno ga odstrani. Pogled v zadnjih 10 minut omogoča storitvi:

- da so zbrane informacije o skupini instanc stabilne,
- preprečiti obnašanje, v katerem storitev neprekinjeno dodaja in odstranjuje vire,
- da varno odstrani instance.

Storitev bo iz skupine instanc odstranila virtualne stroje takrat, ko zazna, da bi zmanjšana skupina instanc zmogla obdelati obremenitev v zadnjih

10 minutah. Te nastavitve se ne da spreminjati. Zakasnitev 10-ih minut zagotavlja, da če je dodan nov virtualni stroj v skupino instanc, se ta izvaja vsaj 10 minut, predno ga lahko storitev ugasne. Razlog za tako delovanje tiči v načinu obračunavanja za virtualne stroje, saj oblak GCP obračunava za minimalno 10 minut.

3.2.2 Google Cloud SQL

Google Cloud SQL je Googlova storitev za podatkovno bazo MySQL. Podpira vse funkcionalnosti podatkovne baze MySQL z nekaj dodatnimi lastnostmi in nekaj omejitvami. Storitev je enostavna za uporabo, ne potrebuje nameščanja in upravljanja ter je idealna za majhne in srednje velike aplikacije.

Storitev podpira namestitveni model nadrejeni-podrejeni za podatkovne baze MySQL, kjer imamo eno glavno podatkovno bazo za bralne in pisalne operacije ter več podatkovnih baz samo za bralne operacije.

Plačevanje za storitev je možno *po porabi* ali *paketno*. Plačevanje po porabi pomeni, da uporabnik plača za vsako minuto, ko se podatkovna baza izvaja, paketno pa pomeni, da uporabnik plača znesek, ki se obračunava dnevno neglede na to, koliko časa se podatkovna baza izvaja. Paketno plačevanje je bolj primerno za uporabnike, ki podatkovno bazo izvajajo več kot 450 ur na mesec, po porabi pa za tiste, ki jo manj kot 450 ur na mesec.

Storitev Google Cloud SQL ponuja 7 možnosti izbire za zmogljivost podatkovne baze. V tabeli 3.1 so zbrane možnosti za model plačevanja po porabi, ki smo ga uporabili pri izdelavi magistrske naloge.

Vir	Cena na uro	Max št. sočasnih povezav
DO (0.125 GB RAM)	$ 0.025	250
D1 (0.5 GB RAM)	$ 0.10	250
D2 (1 GB RAM)	$ 0.19	250
D4 (2 GB RAM)	$ 0.29	500
D8 (4 GB RAM)	$ 0.58	1000
D16 (8 GB RAM)	$ 1.16	2000
D32 (16 GB RAM)	$ 2.31	4000

Tabela 3.1: Tabela tipov instanc za podatkovno bazo na storitvi Cloud SQL s cenami in maksimalnim številom sočasnih povezav.

Poglavje 4

Standard TPC-W

V tem poglavju opišemo ključne dele standarda, ki smo jih uporabili pri implementaciji spletne trgovine in pri razvoju programa za obremenitveno testiranje. Standard TPC-W služi kot izhodišče za obremenitveno testiranje, zato nekaj podrobnosti, ki jih opisuje standard, izpustimo. Standard natančno opiše arhitekturo spletne aplikacije in kako mora biti obremenitev na spletno trgovino izvedena. Standard opiše tudi infrastrukturo, na katero morata biti spletna aplikacija in program za obremenitveno testiranje nameščena.

4.1 Osnovni pojmi obremenitvenega testiranja

Programska oprema v poslovnem svetu postaja vse bolj kompleksna. Moderna programska oprema je sestavljena iz veliko različnih komponent in uporablja različne storitve, kjer vsaka predstavlja potencialno točko okvare. Mobilne in oblačne platforme ter hibridna okolja predstavljajo različne izzive, s katerimi se morajo načrtovalci programske opreme spopadati. Programska oprema mora zato biti dobro stestirana, predno jo namestimo v produkcijsko okolje in ponudimo končnim uporabnikom za uporabo [19].

Nekateri cilji zmogljivostnega testiranja vključujejo identifikacijo ozkih grl

v sistemu, ugotavljanje, kateremu ozkemu grlu lahko spremenimo nastavitve za povečano zmogljivost sistema in zbiranje podatkov, na podlagi katerih se deležniki odločajo o kvaliteti sistema, ki ga testiramo [20].

Obremenitveno testiranje, del zmogljivostnega testiranja, je proces izpostavljanja sistema stopnji dela, ki privede sistem do njegovih omejitev [20]. V literaturi se namesto izraza **obremenitveno testiranje** (angl. *load testing*) uporablja tudi izraz **stresno testiranje** (angl. *stress testing*), vendar gre za dva različna izraza, med katerima je zelo tanka meja. Izraz **obremenitveno testiranje** se uporablja za izvajanje testiranja kapacitete sistema, medtem ko se izraz **stresno testiranje** uporablja za testiranje zmogljivosti in obnašanja sistema do njegove največje kapacitete. Oba izraza sodita pod izraz **zmogljivostnega testiranje** (angl. *performance testing*) [20].

4.2 Opis standarda TPC-W

TPC Benchmark^{TM} W je merilo za transakcijske spletne aplikacije. Obremenitev na transakcijsko spletno aplikacijo je izvedena v nadzorovanem okolju, ki simulira aktivnosti poslovno orientiranega transakcijskega spletnega strežnika. Obremenitev testira komponente sistema v okolju z naslednjimi lastnostmi:

- večkratne spletne brskalniške seje,
- dinamično generiranje strani z dostopom do baze,
- konsistentni spletni objekti,
- sočasno izvajanje različnih zapletenih transakcij,
- podatkovna baza z veliko tabelami različnih velikosti, različnimi atributi in relacijami,
- transakcijska integriteta (lastnosti ACID),
- zagotavljanje dostopa in posodabljanja podatkov.

Metrika za zmogljivost, ki jo uporablja standard TPC-W je: **število spletnih interakcij na sekundo** (angl. *Web Interactions Per Second - WIPS*). Spletne interakcije simulirajo aktivnosti spletne trgovine. Za vsako interakcijo je določena zgornja meja odzivnega časa. Velikost spletne trgovine je določena s številom knjig v podatkovni bazi. Število knjig se giblje med 1 000 in 10 000 000. Metrika za zmogljivost je odvisna od **faktorja skaliranja**, izraženega kot **WIPS@faktor skaliranja**, kjer je faktor skaliranja število knjig v podatkovni bazi.

Standard TPC-W definira tri različne profile z različnim razmerjem operacij brskanja in nakupovanja: primarno nakupovanje (WIPS), primarno brskanje (WIPSb) in primarno naročanje (WIPSo). Vsi rezultati WIPS morajo vsebovati primarno metriko, ki je: stopnja WIPS, cena glede na WIPS ($/WIPS) in čas testiranja.

Standard TPC-W se ne osredotoča na en sam trg, ampak predstavlja splošen model za industrije, kjer želijo promovirati in prodajati produkt ali storitev preko spleta (maloprodaja različnih izdelkov, distribucija programske opreme, rezervacija letalskih vozovnic, itd.). Namen standarda TPC-W ni določanje, kako dejansko implementirati aplikacijo, ampak zmanjšati raznolikost med operacijami, ki jih najdemo v poslovnih spletnih aplikacijah pri tem pa ohranjati ključne performančne lastnosti, izkoriščenost sistema in kompleksnost operacij.

Aplikacija, ki jo predstavlja standard, je spletna knjigarna, ki omogoča uporabnikom brskanje in naročanje knjig. Uporabniki obiščejo prvo stran spletne knjigarne, pregledajo produkte, poiščejo informacije o produktih, izvedejo naročilo ali pa zahtevajo poizvedbo o že izvedenem naročilu. Večina uporabnikovih aktivnosti je osredotočenih okoli brskanja po strani. Nekaj odstotkov vseh obiskov vodi v izvedbo naročila in nakup.

4.2.1 Operacije

Standard TPC-W definira spletno knjigarno s 14 operacijami, ki omogočajo uporabnikom brskanje, naročanje in izvedbo nakupa knjig. Verjetnost opera-

Operacija	Brskanje	Nakupovanje	Naročanje
Domov	29.00%	16.00%	9.12%
Novi produkti	11.00%	5.00%	0.46%
Najboljši avtorji	11.00%	5.00%	0.46%
Pregled artikla	21.00%	17.00%	12.35%
Iskanje	12.00%	20%	14.53%
Rezultati iskanja	11.00%	17.00%	13.08%
Nakupovalna košarica	2.00%	11.60%	13.53%
Registracija uporabnika	0.82%	3.00%	12.86%
Nakup	0.75%	2.60%	12.73%
Rezultati nakupa	0.69%	1.20%	10.18%
Poizvedba o naročilu	0.30%	0.75%	0.25%
Prikaz naročila	0.25%	0.66%	0.22%
Sprememba artikla	0.10%	0.10%	0.12%
Potrditev spremembe artikla	0.09%	0.09%	0.11%

Tabela 4.1: Verjetnost operacij glede na scenarij.

cije je odvisna od scenarija. V standardu TPC-W so definirani trije različni scenariji, in sicer primarno brskanje, primarno nakupovanje in primarno naročanje. Tabela 4.1 prikazuje operacije in njihove verjetnosti za posamezen scenarij.

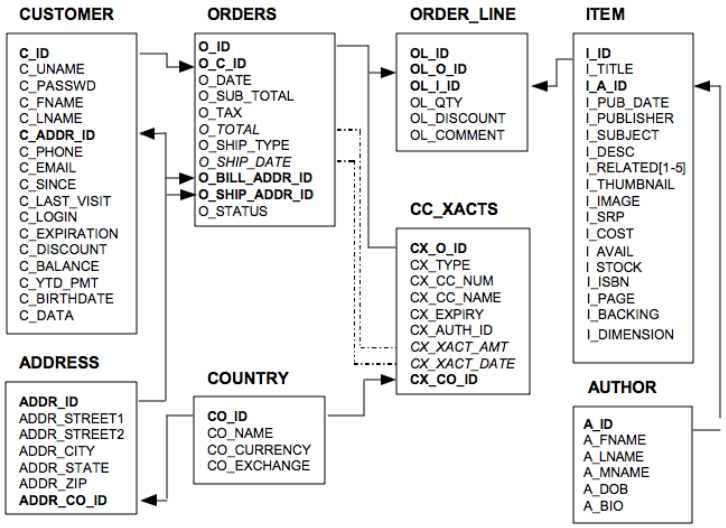
4.2.2 Struktura in velikost podatkovne baze

Standard TPC-W definira shemo podatkovne baze, ki je sestavljena iz 8 tabel in povezave med njimi. Natančno definira tudi tipe in omejitve atributov v vsaki tabeli, ki so na voljo v dokumentu, ki opisuje standard TPC-W [9].

Velikost podatkovne baze je odvisna od *faktorja skaliranja*, ki je določen s številom simuliranih brskalnikov (SB). V tabeli 4.2 so prikazane velikosti vseh tabel po formulah kot jih definira standard TPC-W.

4.2.3 Spletni objekti

Standard TPC-W definira spletne objekte kot slike, ki predstavljajo podatkovni tok različnih medijskih vsebin, kot so: slike, audio, video, animirane



Slika 4.1: Shema podatkovne baze, kot jo določa standard TPC-W.

Ime tabele	Število vrstic
CUSTOMER	2880 * (število SB)
COUNTRY	92
ADDRESS	2 * CUSTOMER
ORDERS	0.9 * CUSTOMER
ORDER_LINE	3 * ORDERS
AUTHOR	0.25 * ITEM
CC_XACTS	1 * ORDERS
ITEM	1k, 10k, 100k, 1M, 10M

Tabela 4.2: Število vrstic v posamezni tabelah v podatkovni bazi.

Velikost slike	Verjetnost
5 K	45 %
10 K	35 %
50 K	13 %
100 K	4 %
250 K	1 %

Tabela 4.3: Verjetnost pojavljanja posamezne velikosti slike.

slike GIF itd. Za vsako sliko definira velikost slike v bajtih in kako pogosto je uporabljena. Velikosti slik so: 5K, 10K, 50K, 100K in 250KB. Končno število unikatnih slik za posamezno velikost je enako številu knjig v podatkovni bazi. Verjetnosti za različne velikosti slik so prikazane v tabeli 4.3.

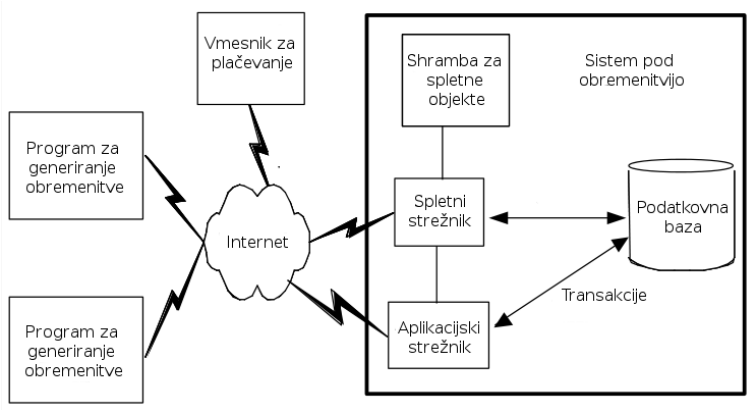
4.2.4 Metrike

Standard TPC-W definira tri različne scenarije za obremenitveni test in za vsakega od njih tudi svojo metriko. Za scenarij nakupovanja definira metriko **število interakcij na sekundo** (angl. *Web interactions per second - WIPS*), za scenarij brskanja **število interakcij na sekundo za brskanje** (WIPSb) in za scenarij naročanja **število interakcij na sekundo za naročanje** (WIPSo). Poleg omenjenih treh metrik definira še metriko $/WIPS, ki določa skupno ceno sistema, ki ga obremenitveno testiramo deljeno s številom spletnih interakcij na sekundo.

4.2.5 Namestitvena arhitektura

Standard TPC-W natančno definira namestitveno arhitekturo spletne aplikacije in programa za obremenitveno testiranje, ki ga standard TPC-W imenuje **oddaljen posnemovalnik brskalnika** (angl. *remote browser emulator*). Namestitvena arhitektura spletne aplikacije za prodajo knjig sestavljajo:

- strežnik za gostovanje spletnih objektov,
- spletni strežnik,



Slika 4.2: Namestitvena arhitektura sistema po standardu TPC-W.

- strežnik za podatkovno bazo,
- aplikacijski strežnik.

Slika 4.2 prikazuje namestitveno arhitekturo spletne aplikacije in komunikacijo preko spleta z enim ali več oddaljenimi posnemovalniki brskalnika ter vmesnik za plačevanje, ki ni del sistema, nad katerim izvajamo obremenitev. Komponente v odebeljenem kvadratku so del sistema, nad katerim izvedemo obremenitev. Za te komponente moramo poznati ceno, če želimo izračunati metriko TPC-W **cena za število spletnih interakcij na sekundo**.

Poglavje 5

Osnovni koncepti vrednotenja oblačnih storitev

V tem poglavju definiramo osnovne koncepte vrednotenja oblačnih storitev, ker je terminologija na področju oblačnih storitev še slabo definirana. Različne skupine na področju IKT (Informacijske in komunikacijske tehnologije, angl. *Information and communication technology (ICT)*) uporabljajo iste izraze za različne pomene, odvisno od domene IKT (Programska oprema, telekomunikacije, proizvodnja) in domenskega jezika. Uporaba dobro definiranega jezika in ključnih konceptov znotraj posamezne domene omogoča skupinam boljšo komunikacijo in tako zmanjša tveganje po nerazumevanju informacij [21].

5.1 Konfiguracija sistema

V kontekstu magistrske naloge pod izrazom **konfiguracija sistema** mislimo eno konkretno namestitev spletne aplikacije v oblak. Pri nameščanju spletne aplikacije lahko nastavljamo množico atributov, npr. število virtualnih strojev, število virtualnih uporabnikov, tip instance za podatkovno bazo, tip instance za spletno trgovino itd. Matematično lahko eno konfiguracijo sis-

tema zapišemo v obliki vektorja:

$$Q = \langle p_Q, f_Q, f_{Q,n}, p_{Q,n}, v_Q, b_Q \rangle \tag{5.1}$$

kjer je:

p_Q - tip instance za podatkovne baze,

f_Q - tip instance za izvajanje spletne aplikacije,

$f_{Q,n}$ - število virtualnih strojev za izvajanje spletne aplikacije tipa f_Q,

$p_{Q,n}$ - število virtualnih strojev za podatkovno baz tipa p_Q,

v_Q - število simuliranih virtualnih uporabnikov,

b_Q - velikost bazena povezav do podatkovne baze.

5.2 Scenarij

Scenarij v kontekstu magistrske naloge predstavlja "recept", ki določa, kako program za obremenitveno testiranje pošilja HTTP zahtevke na spletno aplikacijo. Scenarij je sestavljen po standardu TPC-W, ki določa 14 operacij, med katerimi program za obremenitveno testiranje izbira z vnaprej definirano verjetnostjo. Med dvema zaporednima operacijama vedno poteče določen čas, t. i. čas za razmislek (angl. *think time*), ki je po standardu TPC-W določen na 7 sekund.

5.3 Cilji o ravni storitve

Cilji o ravni storitve (angl. *service level objectives - SLO*) so definirani za vsako izmed 14 operacij, ki jih definira standard TPC-W. SLO nam pove, kakšen je maksimalni sprejemljiv odzivni čas strežnika po tem, ko je program za obremenitveno testiranje nanj poslal zahtevek. Standard TPC-W definira, da mora vsaj 90 % zahtevkov vsake operacije zadovoljiti SLO, da ga smatramo, kot sprejemljivega. V tabeli 5.1 je definiran SLO za vseh 14 operacij, ki jih definira standard TPC-W.

Operacija	Čas v sekundah
Shrani administracijo	20
Zahteva za administracijo	3
Najbolj prodajani produkti	5
Novi produkti	5
Potrdi nakup	5
Zahteva za nakup	3
Registracija uporabnika	3
Domov	3
Prikaži naročilo	3
Poizvedba o naročilu	3
Podrobnosti o izdelku	3
Zahtevek za iskanje	3
Rezultati iskanja	10
Nakupovalna košarica	3

Tabela 5.1: SLO za 14 operacij, ki jih definira standard TPC-W.

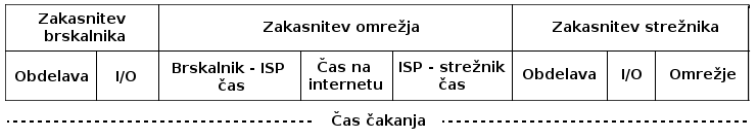
5.4 Kapaciteta

Najbolj splošna definicija kapacitete je:

"kapaciteta je količina, ki jo lahko nekaj drži".

Poznamo veliko različnih definicij kapacitete, odvisno od področja in vprašanja, na katerega želimo odgovoriti s kapaciteto. Kapaciteta je lahko definirana kot:

- količina vode, ki jo lahko kozarec drži, tako da se voda ne zlije čez rob kozarca,
- volumen zraka, ki ga lahko človek naenkrat vdihne,
- število bitov, ki se lahko naenkrat prenesejo čez komunikacijski kanal,
- količina električnega naboja v bateriji, ki lahko proizvede določeno napetost.



Slika 5.1: Odzivni čas po komponentah.

V računalništvu se je v ta namen razvilo področje načrtovanja kapacitet programske opreme in arhitektur. V oblaku je to področje zelo pomembno, saj morajo ponudniki storitev znati predvideti potrebo po računalniških virih in jo ustrezno zaračunavati. Na drugi strani pa morajo načrtovalci programske opreme, ki uporabljajo te storitve in za njih plačujejo, računalniške vire znati cenovno in tehnično učinkovito uporabiti za doseganje potreb svojih uporabnikov in doseganje dobrih poslovnih rezultatov.

5.5 Odzivni čas

Čas, ki ga sistem porabi, da se odzove na uporabnikovo zahtevo, imenujemo **odzivni čas**. Kot primer lahko navedemo čas, ki je potreben, da se spletna stran prikaže v uporabnikovem brskalniku z rezultati iskanja po katalogu. Odzivni čas, ponavadi izmerjen v sekundah, lahko razbijemo na več komponent.

Slika 5.1 prikazuje tri glavne komponente odzivnega časa zahtevka za iskanje na spletni strani za trgovanje: zakasnitev brskalnika, zakasnitev omrežja in zakasnitev strežnika. Zakasnitev brskalnika vključuje čas obdelave in čas vhodno/izhodnih operacij, ki so potrebne za pošiljanje zahtevka in prikazovanje spletne strani za rezultate. Zakasnitev omrežja vključje čas prenosa od brskalnika do ponudnika spletnih storitev (ISP) uporabnika, čas porabljen na spletu in čas porabljen za komunikacijo med ponudnikom spletnih storitev (ISP) in spletno trgovino ter strežnikom, na katerem se izvaja. Tretja komponenta vključuje čase, vključene v obdelavo zahtevka spletne trgovine, vse I/O čase in čas porabljen v omrežju spletne trgovine, ki je interen. Vsaka od

treh komponent vključuje čas, porabljen na čakanje različnih virov (procesorji, diski, omrežje). Ta čas imenujemo *čas čakanja*. Čas čakanja je odvisen od števila zahtevkov, ki jih procesira sistem. Večje je število zahtevkov v sistemu, večji bo čas čakanja.

5.6 Elastičnost

Pojem elastičnosti izvira iz fizike, kjer je elastičnost definirana kot sposobnost materiala, da se povrne v prvotno stanje po deformaciji. Koncept elastičnosti je bil prenešen na računalništvo v oblaku v smislu dodajanja in odstranjevanja računalniških virov, ki je običajno izvedeno avtomatsko [22].

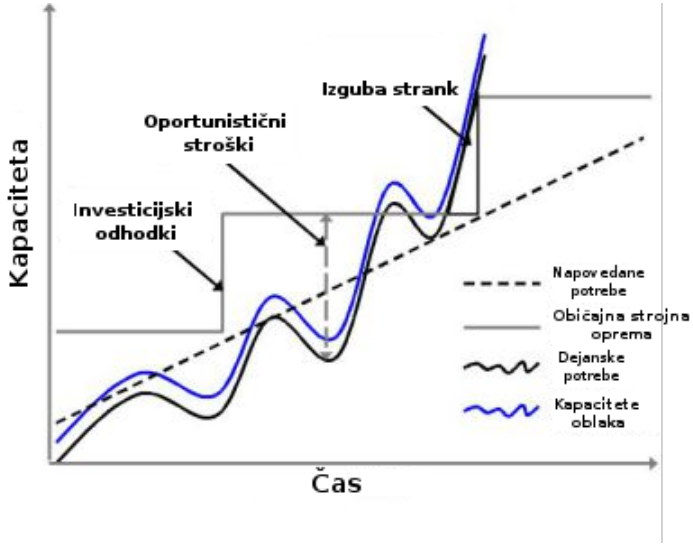
V literaturi najdemo veliko različnih definicij, kjer so si nekatere med seboj celo nasprotujoče. Herbst et. al [23] analizirajo različne definicije, izpostavijo njihove pomankljivosti in ponudijo svojo, povzeto definicijo elastičnosti, ki jo tudi uporabljamo v magistrski nalogi:

> *"Elastičnost je stopnja, s katero se je sistem sposoben prilagoditi spremembam v obremenitvi z avtomatskim dodajanjem in odstranjevanjem virov tako, da so viri v vsakem trenutku prilagojeni trenutnim zahtevam obremenitve, kolikor je to mogoče."*

Ogromno truda je vloženega v razvoj elastičnosti, da se sistem kar se da najbolje prilega trenutni obremenitvi, vendar se ponudniki oblaka težko izognejo pretirani oskrbi (angl. *over provisioning*) in premajhni oskrbi (angl. *under provisioning*) z računalniškimi viri (slika 5.2). Pretirano oskrbovanje lahko ponudniku oblaka povzroči preveliko stroškov, premajhna oskrba pa izgubo denarja in slabo kvaliteto storitev.

5.7 Skalabilnost

Skalabilnost aplikacij je dandanes ključnega pomena za uspešno poslovanje spletnih podjetij, kot so Facebook, Instagram in Snapchat. Njihove aplikacije



Slika 5.2: Pretirana oskrba in premajhna oskrba značilna za oblak.

so tako obiskane, da že najmanjši izpad storitve lahko pomeni milijonsko izgubo. Omenjena podjetja si ne morejo privoščiti, da njihova aplikacija ne bi delovala niti za enega uporabnika.

Definicijo koncepta skalabilnosti povzamemo po definiciji skalabilnosti, kot so jo definirali Lehrig et al. [24] z metodo sistematičnega pregleda literature:

> *"Skalabilnost je sposobnost oblačnega sloja, da poveča svoje kapacitete z razširjanjem količine uporabljenih storitev na nižjem nivoju"*

Računalniški viri, ki podpirajo delovanje aplikacije, se lahko ob povečani obremenitvi dodajajo ročno ali avtomatsko. Kapaciteto računalniških virov lahko povečamo s povečevanjem zmogljivosti procesorskega, pomnilniškega, podatkovnega ali katerega drugega vira. Takemu povečevanju kapacitete pravimo **vertikalno skaliranje** oz. skaliranje navzgor. Tak način skaliranja je lahko samo ročen in je zelo nepraktičen, saj moramo v tem primeru delovanje računalnika (virtualnega ali fizičnega) zaustaviti.

Poznamo tudi **horizontalno skaliranje**, kjer ne povečujemo zmogljivosti računalniških virov, ampak jih dodajamo. Tako skaliranje je lahko avtomatsko, saj prejšnih računalniških virov ni potrebno zaustavljati in lahko med dodajanjem novih nemoteno delujejo.

Povečevanje računalniških virov včasih ni dovolj, če je arhitektura aplikacije slabo zasnovana in ne zna izkoristiti dodatnih računalniških virov, zato morajo biti tudi aplikacije ustrezno arhitekturno zasnovane, da je skalabilnost aplikacije sploh možna.

Poglavje 6

Predlagane metrike

V tem poglavju so predstavljene metrike za merjenje uspešnosti obremenitvenega testiranja, ki smo jih definirali s pomočjo metrik, definiranih v standardu TPC-W in preučevanjem literature, povezane s skalabilnostjo [10] [11].

Metrike pomagajo pri ustvarjanju mnenja o kvaliteti storitve. Metrika mora biti dovolj splošna, objektivna, enako merjena in enako ovrednotena za različne storitve, tako da lahko njihove vrednosti med seboj primerjamo [25].

Nove metrike predlagamo, ker metrike iz standarda TPC-W niso primerne za merjenje zmogljivosti aplikacije, ki se izvaja v oblaku, ker ne upoštevajo dinamičnosti skalabilnega in elastičnega okolja oblaka ter modela plačevanja po porabi. Poleg tega metrike iz standarda TPC-W testirajo samo fiksne namestitvene konfiguracije sistemov, ne pa tudi sistemov nameščenih z vklopljenim avtomatskim skaliranjem.

Metrike definiramo v dveh korakih. Prvi korak je definicija osnovnih metrik, ki opisujejo lastnosti oblaka in aplikacije, ki jih merimo z obremenitvenim testom. Iz osnovnih metrik nato izpeljemo še dve novi metriki, ki ju uporabljamo v meritvah in za izdelavo stroškovnega modela.

6.1 Osnovne metrike

Osnovne metrike definiramo na podlagi lastnosti oblaka. Definirane so tiste lastnosti, katerih veličine lahko merimo s programom za obremenitveno testiranje, ali pa so nam na voljo kot del storitve javnega oblaka.

Število virtualnih uporabnikov je število sočasnih uporabnikov, ki jih simulira program za obremenitveno testiranje, tako da pošilja HTTP zahtevke na spletno aplikacijo po vnaprej definiranem scenariju.

Odzivni čas je čas, ki preteče med pošiljanjem zahtevka in odgovorom strežnika za prikaz spletne strani. Zahtevek pošlje program za obremenitveno testiranje. Podrobenje je odzivni čas opisan v razdelku 5.5.

Število virtualnih strojev je maksimalno število virtualnih strojev v oblaku, ki so bili med testiranjem na voljo za procesiranje zahtevkov. Število virtualnih strojev se lahko med izvajanjem testiranja povečuje/zmanjšuje ročno ali avtomatsko.

Cena je vsota v evrih, ki nam jo zaračuna ponudnik javnega oblaka za uporabo storitev in virov za izvajanje aplikacije v času testiranja. Cena ne vključuje virov, ki so potrebni za izvajanje programa za obremenitveno testiranje, ampak vključjuje samo vire za izvajanje aplikacije in njene obremenitve.

6.2 Izpeljane metrike

Izpeljane metrike definiramo na podlagi osnovnih metrik. Definiramo dve metriki, za kateri menimo, da sta dovolj splošni, da se jih lahko neodvisno uporabi tudi za testiranje drugačnih aplikacij, kot je naša, in za druge oblake.

6.2.1 Kapaciteta

Definicija **kapacitete** izhaja iz metrike WIPS iz standarda TPC-W in jo implicitno vključuje. Metrika WIPS meri število interakcij na sekundo, kapaciteta pa meri število virtualnih uporabnikov, ki izvajajo interakcije s spletno aplikacijo. Prav tako kot metrika WIPS tudi naša definicija metrike *kapaciteta* zahteva definicijo *ciljev glede ravni storitve* za vsako operacijo, ki jo testiramo. **Kapaciteto** definiramo kot:

"*Kapaciteta je število virtualnih uporabnikov, ki jih merjena konfiguracija sistema lahko podpre brez kršitev SLO.*"

Matematično lahko metriko **kapaciteta** zapišemo kot funkcijo:

$$c(Q_p, SLO) = k_{p,Q} \tag{6.1}$$

kjer je:

Q_p - ena konfiguracija sistema za oblak p,

SLO - matrika, kjer vrstice ustrezajo operaciji, stolpci pa maksimalnemu odzivnemu času v sekundah,

$k_{Q,p}$ - kapaciteta za konfiguracijo sistema Q za oblak p.

6.2.2 Skalabilnost

Pri definiciji metrike **skalabilnost** izhajamo iz definicije skalabilnosti, kot so jo definirali Lehrig et al. [24] s sistematičnim pregledom literature:

"*Skalabilnost je sposobnost oblačnega sloja, da poveča svoje kapacitete z razširjanjem količine uporabljenih storitev na nižjem nivoju*"

Zgornjo definicijo skalabilnosti apliciramo in povežemo z metriko *kapaciteta* (6.1). Skalabilnost predstavimo s funkcijo:

$$s(Q_p) = k_{p,Q} \tag{6.2}$$

kjer je:

Q_p - ena konfiguracija sistema za oblak p,

$k_{Q,p}$ - kapaciteta za konfiguracijo sistema Q za oblak p.

Metrika **skalabilnost** opiše, kako se kapaciteta $k_{p,Q}$ spreminja z različnimi konfiguracijami sistema Q_p. **Skalabilnost** definiramo kot:

"*Skalabilnost je sposobnost sistema, da podpre naraščajoče zahteve po kapaciteti z ročnim ali avtomatskim skaliranjem.*"

Poglavje 7

Razvita orodja in aplikacije

V tem poglavju predstavimo aplikacije in orodja, ki smo jih razvili in uporabili za obremenitveno testiranje. Spletno trgovino, namestitvene programe in program za generiranje obremenitve smo zasnovali, tako da je vsaka od naštetih aplikacij neodvisna ena od druge. Uporabnik lahko uporabi samo eno aplikacijo od naštetih, ostale pa zamenja s svojimi.

Spletna aplikacija z grafičnim uporabniškim vmesnikom olajša uporabo vseh prej naštetih aplikacij, ki sicer ne vsebujejo grafičnega uporabniškega vmesnika.

7.1 Spletna trgovina

Spletna trgovina predstavlja vstopno točko za obremenitveni test. Spletno trgovino smo razvili po zgledu obstoječe implementacije standarda TPC-W v JavaEE 1.5, ki smo jo našli na spletu [26]. V arhivu z izvorno kodo smo našli tudi program za generiranje slik, ki generira slike takšnih velikosti, kot je določeno v tabeli 4.3.

Spletno trgovino smo modernizirali, tako da smo jo na novo implementirali v programskem jeziku Java z uporabo SpringFramework razvojnega ogrodja. Za transakcije in dostop do podatkov v podatkovni bazi smo uporabili Hibernate objektno relacijsko preslikavo (angl. *Object-relational mapping*

Standard TPC-W	AWS	GCP
Podatkovna baza	RDS	Cloud SQL
Shramba za spletne objekte	S3	Cloud Storage
Spletni strežnik	Nginx na EC2	Nginx na GCE
Aplikacijski strežnik	Tomcat na EC2	Tomcat na GCE

Tabela 7.1: Preslikava TPC-W arhitekture na AWS in GCP oblak.

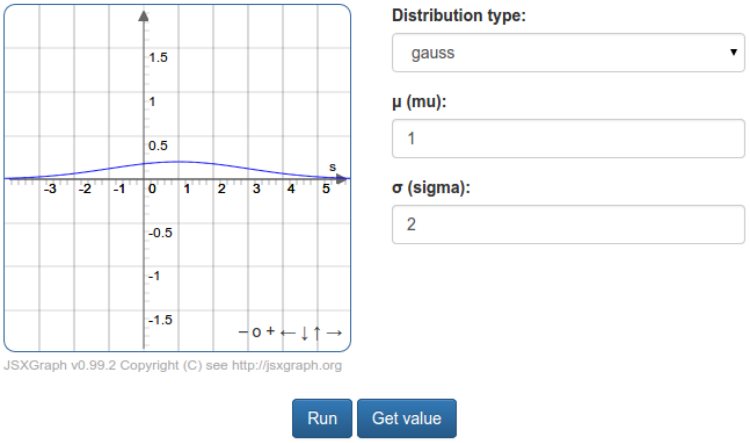
- *ORM*), ki omogoča enoten vmesnik za dostop do različnih relacijskih podatkovnih baz. Za upravljanje z bazenom povezav do podatkovne baze smo uporabili c3p0 [27] knjižnico. Za zgraditev paketa WAR, ki se izvaja na aplikacijskem strežniku Tomcat, smo uporabili sistem za grajenje programske opreme Maven.

Pri nameščanju spletne trgovine v izbrana oblaka, smo upoštevali arhitekturo, kot jo definira standard TPC-W (4.2). Standard TPC-W definira, da spletna trgovina uporablja strežnik za podatkovno bazo, strežnik za shranjevanje spletnih objektov, spletni strežnik in aplikacijski strežnik. Tabela 7.1 prikazuje, kako smo omenjeno arhitekturo preslikali na oblaka AWS in GCP ter katere storitve smo za to uporabili.

Programski opremi Nginx in Tomcat se pri obeh oblakih izvajata na istem virtualnem stroju. Nginx je v našem primeru nastavljen kot posredniški strežnik za dostop do aplikacijskega strežnika Tomcat, saj dobra praksa narekuje, da se aplikacijskega strežnika ne izpostavlja direktno na splet.

7.2 Vmesnik za plačevanje

Vmesnik za plačevanje smo razvili v programskem jeziku Python. Razvili smo ga kot majhno spletno aplikacijo s preprostim uporabniškim vmesnikom (Slika 7.1) in programskim vmesnikom REST (angl. *Representational State Transfer*), ki ga uporabljamo v spletni trgovini. Vmesnik za plačevanje ne izvede dejanskega plačila, ampak ga simulira, tako da po izbrani distribuciji generira odzivni čas. Generiranje odzivnega časa je možno po naslednjih



Slika 7.1: Zaslonska slika uporabniškega vmesnika za vmesnik za plačevanje.

distribucijah:

- Gausova,
- eksponentna,
- gama,
- logaritemska,
- Pareto,
- Weibull,
- konstantna.

Vsaka distribucija ima enega ali več parametrov, s katerimi spreminjamo njeno porazdelitev. Kljub temu smo vsaki distribuciji dodali še parameter k, ki premakne končno vrednost distribucije za k mest po osi x.

Vmesnik za plačevanje smo namestili na oblak Heroku, ki ponuja okolje za gostovanje spletnih aplikacij. Vmesnik za plačevanje je na voljo na spletnem naslovu [28].

7.3 Namestitveni programi

Namestitvene programe smo razvili zaradi lažjega nameščanja spletne trgovine na oba oblaka, saj je drugače potrebno veliko ročnega dela za namestitev ene konfiguracije. Namestitveni programi nam pomagajo tudi pri preprečevanju napak, do katerih lahko pride zaradi nepozornosti in nenatančnosti pri ročnem nameščanju.

Namestitveni koraki se med oblakoma malenkost razlikujejo zaradi različnega programskega vmesnika API in delovanja oblaka. Pri obeh oblakih namestitveni programi eno konfiguracijo namestijo v dveh korakih, in sicer: najprej podatkovno bazo, nato še spletno trgovino. Korak nameščanja spletne trgovine je nato razdeljen še na koraka za vzpostavljanje infrastrukture za spletno trgovino in nameščanje spletne trgovine. Korak nameščanja spletne trgovine je za oba oblaka enak.

Zaradi hitrejšega oskrbovanja z virtualnimi stroji smo za oba oblaka pripravili sliko virtualnega stroja z nameščeno in nastavljeno programsko opremo (Tomcat, Nginx), tako da ni potrebno tega početi pri vsakem oskrbovanju z virtualnim strojem. V nadaljevanju podrobneje opišemo posamezne korake, ki so potrebni za nameščanje spletne trgovine v izbran oblak.

7.3.1 Oskrbovanje in nastavljanje podatkovne baze

Za oskrbovanje in nastavljanje podatkovne baze je potrebno opraviti določeno zaporedje klicev programskega vmesnika izbranega oblaka. V nadaljevanju opišemo korake, ki so potrebni za oskrbovanje in nastavljanje podatkovne baze, kjer vsak korak ustreza enemu klicu programskega vmesnika.
Pri oblaku AWS so potrebni naslednji koraki:

1. Ustvari *varnostno skupino* za dostop do podatkovne baze MySQL.
2. Ustvari virtualni stroj s podatkovno bazo MySQL z uporabo storitve RDS.

3. Uvozi podatkovno bazo.
4. Shrani naslov za dostop do podatkovne baze.

Pri oblaku GCP so potrebni naslednji koraki:

1. Ustvari virtualni stroj s podatkovno bazo MySQL z uporabo storitve Cloud SQL.
2. Ustvari uporabnika za povezovanje na podatkovno bazo.
3. Ustvari podatkovno bazo.
4. Uvozi podatkovno bazo.
5. Shrani naslov za dostop do podatkovne baze.

7.3.2 Nameščanje spletne trgovine

Nameščanje spletne trgovine je razdeljeno na dva koraka. Prvi korak je vzpostavitev infrastrukture za spletno trgovino, drugi pa nameščanje spletne trgovine. V primeru avtomatskega skaliranja je korak nameščanja spletne trgovine vključen v korak vzpostavitve infrastrukture, v nasprotnem primeru pa je to zadnji korak pri nameščanju spletne trgovine.

Vzpostavljanje infrastrukture za spletno trgovino

Vzpostavljanje infrastrukture za spletno trgovino na oblaku **AWS** poteka po naslednjih korakih:

1. Ustvari par ključev za avtentikacijo in dostop do virtualnega stroja.
2. Če ni zahtevana samodejna skalabilnost, potem:
   (a) ustvari virtualne stroje iz slike virtualnega stroja AMI,
   (b) ustvari izenačevalec obremenitve, če je število virtualnih strojev > 1,
   (c) namesti spletno trgovino na virtualne stroje (7.3.2).

3. Če je zahtevana samodejna skalabilnost, potem:
   (a) ustvari virtualni stroj iz slike virtualnega stroja AMI,
   (b) namesti spletno trgovino na virtualni stroj (7.3.2),
   (c) ustvari sliko virtualnega stroja,
   (d) ustvari izenačevalec obremenitve,
   (e) ustvari varnostno skupino za SSH in HTTP dostop,
   (f) ustvari *zagonske nastavitve*,
   (g) ustvari *skupino za avtomatsko skaliranje*,
   (h) ustvari politiko skaliranja (angl. *scaling policy*),
   (i) ustvari alarme na storitvi CloudWatch.

Vzpostavljanje infrastrukture za spletno trgovino na oblaku GCP poteka po naslednjih korakih:

1. Če ni zahtevana samodejna skalabilnost, potem:
   (a) Ustvari virtualne stroje iz slike virtualnega stroja AMI.
   (b) Ustvari *izenačevalec obremenitve*, če je število virtualnih strojev > 1:
      - rezerviraj nov statični naslov IP,
      - ustvari nov *pregled stanja* (3.2.1),
      - ustvari nov *ciljni bazen* (3.2.1),
      - ustvari novo *pravilo posredovanja* (3.2.1).
   (c) Namesti spletno trgovino na virtualne stroje (7.3.2)
2. Če je zahtevana samodejna skalabilnost, potem:
   (a) ustvari nov virtualni stroj,
   (b) nastavi možnost, da se podatkovni disk po zaustavitvi virtualnega stroja ne izbriše,

(c) namesti spletno trgovino na virtualni stroj (7.3.2),

(d) izbriši virtualni stroj (podatkovni disk se ne izbriše),

(e) ustvari sliko virtualnega stroja iz podatkovnega diska,

(f) ustvari *predlogo instanc* (3.2.1),

(g) ustvari *skupino instanc* (3.2.1),

(h) ustvari *autoscaler* iz *skupine instanc* (3.2.1),

(i) ustvari *izenačevalec obremenitve HTTP zahtev* (3.2.1):

- ustvari *pregled stanja*,
- ustvari *zaledno storitev* iz *pregleda stanja* in *skupine instanc*,
- ustvari *preslikavo naslova URL* iz *zaledne storitve*,
- ustvari *ciljni posredovalnik HTTP zahtevkov* iz *preslikave naslova URL*,
- rezerviraj nov statični naslov IP,
- ustvari *globalno posredovalno pravilo*,

Nameščanje spletne trgovine na virtualni stroj

Nameščanje spletne trgovine je za oba oblaka enako, saj program za nameščanje spletne trgovine na virtualni stroj potrebuje samo naslov IP virtualnega stroja in omogočen SSH dostop. Nameščanje vključuje naslednje korake:

1. poveži se z virtualnim strojem preko SSH povezave,
2. pridobi spletno trgovino z naslova `https://storage.googleapis.com/magistrska/showcaseV3-sql.war`,
3. odarhiviraj pridobljeno datoteko `showcase-sql.war`,
4. zapiši podatke za dostop do baze v `app.properties` datoteko,
5. zamenjaj *${connection_pool_size}* označbo mesta (angl. *placeholder*) v `hibernate.xml`, datoteki z vrednostjo iz nastavitvene datoteke

6. kopiraj spletno trgovino v Tomcat namestitveno mapo,

7. ponovno zaženi strežnik Tomcat.

7.3.3 Organiziranost izvorne kode

Namestitvene programe smo razvili v programskem jeziku Python z uporabo programskega ogrodja `boto` za dostop do storitev na oblaku AWS in uporabo knjižnice `google-api-python-client` za dostop do storitev oblaka GCP.

Izvorna koda za namestitvene programe je organizirana v tri sloje in sovpada razdelitvi oblaka na tri glavne sloje (IaaS, PaaS, SaaS):

- *infrastructure* (slo. infrastruktura),
- *platform* (slo. platforma),
- *software* (slo. programska oprema).

Pod nivo **infrastruktura** sodijo programi za oskrbovanje z virtualnimi stroji, ustvarjanje izenačevalcev obremenitve, ustvarjanje slike virtualnega stroja, dodeljevanje ključev virtualnemu stroju in ustvarjanje varnostnih skupin za kontrolo dostopa. Pod nivo **platforma** sodijo programi, ki uporabljajo storitve PaaS nivoja izbranega oblaka, npr. uporaba Amazonove storitve za relacijske podatkovne baze RDS ali uporaba storitve Cloud SQL za podatkovne baze MySQL. Pod nivo **programska oprema** sodijo programi za nastavljanje in nameščanje programske opreme na virtualni stroj, za nalaganje spletne trgovine na virtualni stroj in nastavljanje spletne trgovine.

Programi so zapakirani v Python paket, tako da jih lahko uporabnik namesti na svoj računalnik in vključi v svoje programe ter razširi funkcionalnosti. Na voljo so na GitHub-u [29] za prenos.

Namestitvenim programom smo priložili tudi zagonski program, ki je v datoteki `bin/run.py`, ki uporabniku omogoča lažje zaganjanje namestitvenih programov. Programom so priloženi tudi vzorci nastavitvene datoteke, ki jo mora uporabnik uporabiti pri zagonu. Datoteko mora pred zagonom urediti

in nastaviti vrednosti, ki so potrebne za uspešno povezovanje z izbranim oblakom.

7.3.4 Uporaba

Namestitveni programi kot vhodni parameter sprejmejo tip oblaka in pot do nastavitvene datoteke. Nastavitve v nastavitveni datoteki se razlikujejo med oblaki in so podrobneje opisane na spletni strani GitHub [29]. Spodnji primer namesti spletno trgovino na oblak AWS:

```
$ python run.py aws config.aws.ini
```

Nastavitev vseh potrebnih storitev in namestitev programske opreme za konfiguracijo z enim virtualnim strojem za spletno trgovino in eno podatkovno bazo traja približno 20 min. Če povečamo število virtualnih strojev, ali velikost podatkovne baze, ali število virtualnih strojev za podatkovno bazo, se poveča tudi čas namestitve.

Po končani namestitvi se uporabniku na zaslonu izpiše naslov URL do virtualnega stroja ali izenačevalca obremenitve, kamor je nameščena spletna trgovina.

7.4 Program za generiranje obremenitve

Program za generiranje obremenitve smo razvili, zato da avtomatiziramo proces nameščanja in distribuirano izvajanje orodja JMeter v izbranem oblaku ter avtomatske obdelave in vizualizacije rezultatov testiranja. Proces nameščanja in izvajanja orodja JMeter mora biti pravilno in natančno izveden, da se test uspešno izvede. Naloga programa za generiranje obremenitve ni samo nameščanje orodja JMeter v izbran oblak in izvajanje obremenitve, ampak tudi zbiranje, obdelava in vizualizacija podatkov. S pomočjo obdelave podatkov in vizualizacije lahko nato dobljene rezultate ovrednotimo z definiranimi metrikami. Program za generiranje lahko logično razdelimo na dve komponenti:

- nameščanje in izvajanje obremenitve iz izbranega oblaka,
- zbiranje, obdelava in vizualizacija podatkov.

7.4.1 Nameščanje in izvajanje obremenitve iz izbranega oblaka

Program, za generiranje obremenitve uporablja program JMeter, ki je podrobneje opisan v razdelku 7.4.2. Za nameščanje orodja JMeter v oblak AWS smo uporabili programsko ogrodje `boto`, za nameščanje v oblak GCP pa
knjižnico `google-api-python-client`.

Tako kot namestitveni programi tudi program za generiranje obremenitve uporablja nastavitveno datoteko za vsak oblak posebej, v kateri so podatki za povezovanje na izbran oblak in podatki za generiranje obremenitve. Program sprejme dva vhodna parametra, ime oblaka, s katerega želimo, da se izvede obremenitev in pot do nastavitvene datoteke, v kateri je tudi informacija o naslovu URL spletne aplikacije za testiranje. Primer zagona za oblak AWS:

```
$ python run.py aws config.aws.ini
```

Proces nameščanja programa za obremenitveno testiranje na izbran oblak je podoben, kot pri nameščanju spletne trgovine v izbran oblak (7.3.2):

- ustvari virtualni stroj,
- omogoči SSH dostop na virtualni stroj,
- poveži se na virtualni stroj preko SSH povezave,
- pridobi JMeter z naslova `https://storage.googleapis.com/magistrska/jmeter_master.tar.gz`,
- odarhiviraj datoteko `jmeter_master.tar.gz`,
- prenesi JMeter scenarij na virtualni stroj.

Program JMeter na vsakem virtualnem stroju zaženemo šele po tem, ko smo ga namestili na vse virtualne stroje, zato da se vsi programi JMeter zaženejo ob istem času. Program JMeter zaženemo brez uporabniškega vmesnika, mu podamo pot do scenarija, povemo, v katero datoteko naj shranjuje dnevnik zapisov izvajanja testa in podamo naslov URL, nad katerim naj izvede obremenitev. Celoten ukaz, ki ga izvedemo na virtualnem stroju, je:

```
$ jmeter -n -t scenario.jmx -j scenario.log -Jhost=<URL naslov>
```

Zadnji korak pri nameščanju in izvajanju obremenitve je zbiranje, obdelava in vizualizacija podatkov. Program z oblaka, v katerem se izvaja, zbere podatke o izkoriščenosti CPU virtualnih strojev, na katerih se izvaja spletna trgovina in izkoriščenosti CPU podatkovne baze (možno samo za storitev RDS). Podatki, na katerih virtualnih strojih se izvaja spletna trgovina, program najde v nastavitveni datoteki.

7.4.2 Orodje JMeter

Program JMeter je odprtokodna aplikacija, namenjena za testiranje in merjenje zmogljivostnega in funkcionalnega obnašanja aplikacij tipa strežnik-odjemalec. Program JMeter ima vlogo odjemalca v aplikaciji tipa strežnik-odjemalec. Z njim lahko zmogljivostno merimo statične in dinamične vire, kot so:

- spletne storitve,
- spletne strani,
- podatkovne baze,
- strežnike FTP.

Program JMeter je napisan v programskem jeziku Java, kar mu omogoča neodvisnost od platforme, na kateri se izvaja in enostavno razširitev njegovih funkcionalnosti preko programskega vmesnika [20]. Testiranje lahko

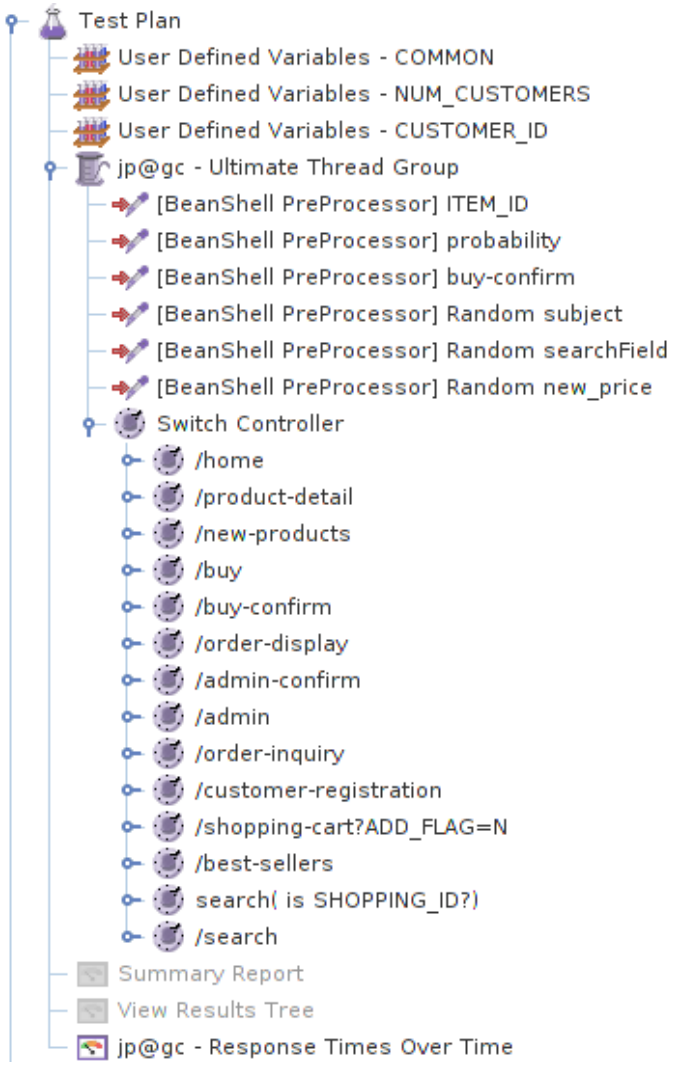
izvedemo ali iz namiznega grafičnega uporabniškega vmesnika ali pa iz ukazne vrstice. Testiranje je lahko izvedeno v razmerju nadrejeni-podrejeni, vendar tega načina za izdelavo magistrske naloge nismo uporabili, ker pri velikem številu podrejenih programov JMeter porabi zelo veliko pomnilnika in procesorske moči.

Program JMeter deluje tako, da simulira uporabnike, ki pošiljajo zahteve na strežnik, ki ga testiramo, po vnaprej pripravljenem scenariju. Na koncu simulacije ponuja različne statistike o testu v obliki tabel in grafov. Uporabnik, s pomočjo namizne aplikacije, ki vsebuje različne gradnike, s katerimi lahko uporabnik ustvari scenarij, podpira različne funkcionalnosti za testiranje ciljnega strežnika. Scenarij je predstavljen kot drevo (slika 7.2), v fizični obliki pa je shranjen kot XML datoteka. Program JMeter v osnovni različici omogoča samo izvajanje linearnega scenarija, ne pa tudi bolj zapletenih scenarijev, zato smo vanj vključili zbirko vtičnikov, ki to omogočajo.

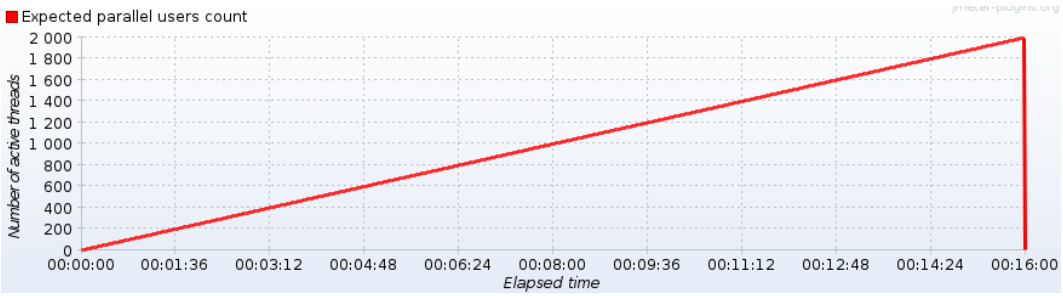
Program JMeter podpira tudi izvajanje programov Beanshell med izvajanjem scenarija. Beanshell je interpreter za programski jezik Java. V orodju JMeter smo uporabili Beanshell za izbiro operacije, ki naj jo izvede JMeter po verjetnosti, kot je določena v standardu TPC-W.

S preizkušanjem smo ugotovili, da en program JMeter lahko izvede 2000 virtualnih uporabnikov na računalniku z najmanj 4GB pomnilnika. Za generiranje obremenitve smo pripravili 16 minutni linearni scenarij (slika 7.3), ki simulira nenadno povečanje obiska spletne aplikacije. Scenarij smo nastavili tako, da beleži podatke o testu v CSV datoteko `response-time-over-time.csv`. V omenjeno datoteko orodje JMeter za vsak zahtevek, ki ga pošlje na strežnik zabeleži:

- časovni žig (angl. *timestamp*) odgovora na zahtevek,
- odzivni čas,
- statusna koda odziva,
- ime operacije,



Slika 7.2: Scenarij v programu JMeter za obremenitveno testiranje spletne trgovine.



Slika 7.3: Primer scenarija, ki v 16 minutah simulira 2000 virtualnih uporabnikov.

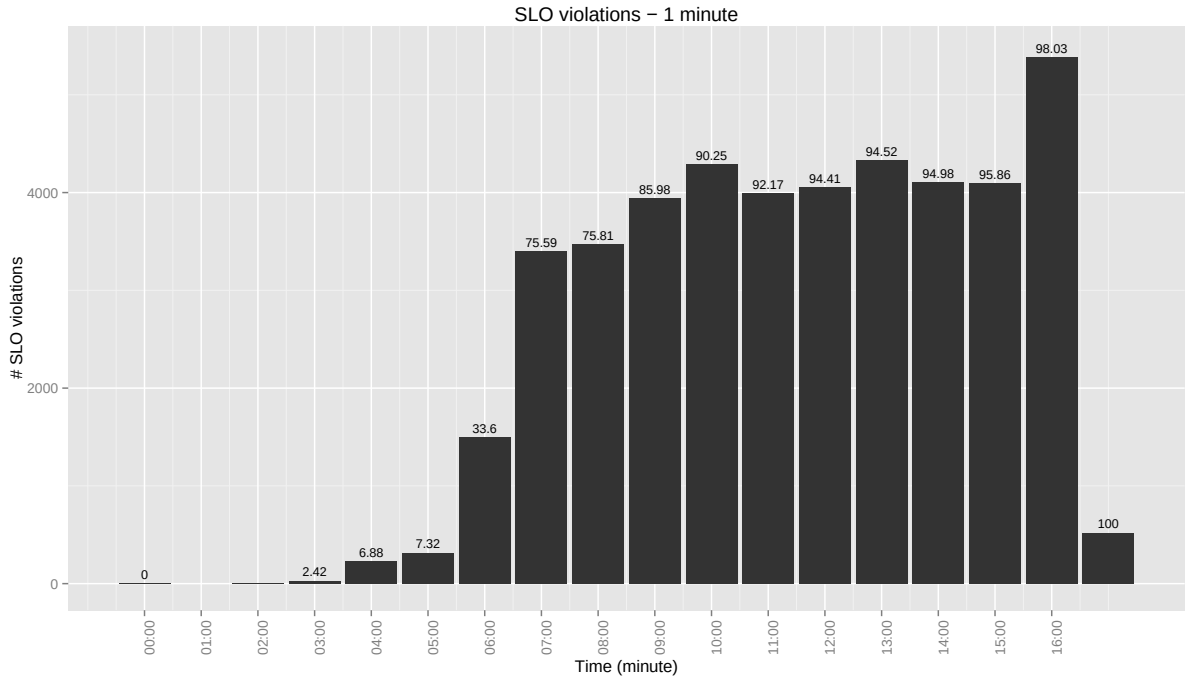
- sporočilo odziva,
- identifikacijska številka niti.

Datoteke nato zberemo z vseh programov JMeter, jih združimo v eno in pošljemo programu za obdelavo in vizualizacijo podatkov.

7.4.3 Obdelava in vizualizacija podatkov

Obdelavo podatkov iz datoteke `response-time-over-time.csv` izvedemo v programskem jeziku Python, kjer podatke obdelamo, tako da zapise iz omenjene datoteke združimo po minutnih časovnih intervalih glede na časovni žig. Agregirane podatke skupaj s podatki o izkoriščenosti CPU virtualnih strojev za spletno aplikacijo in podatki o izkoriščenosti CPU podatkovne baze, nato posredujemo programu R, ki je program za statistično obdelavo podatkov. V programu R izračunamo in vizualiziramo:

- število kršitev SLO v odstotkih v vsaki minuti (slika 7.4),
- izračunamo in vizualiziramo povprečno izkoriščenost CPU za virtualne stroje za spletno aplikacijo (slika 7.5),
- vizualiziramo izkoriščenost CPU podatkovne baze - možno samo za storitev RDS (slika 7.6),

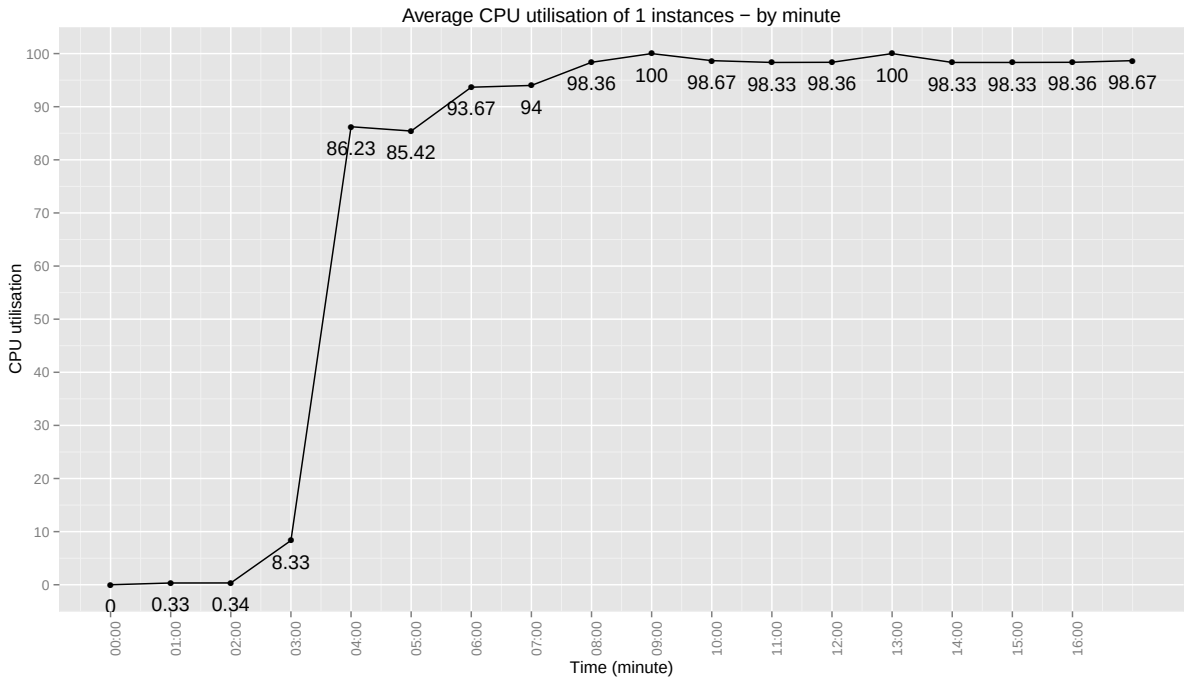


Slika 7.4: Odstotek kršitev SLO po minutah za meritev z eno podatkovno bazo tipa db.m3.medium, eno instanco tipa m3.medium za spletno aplikacijo in 2000 virtualnih uporabnikov.

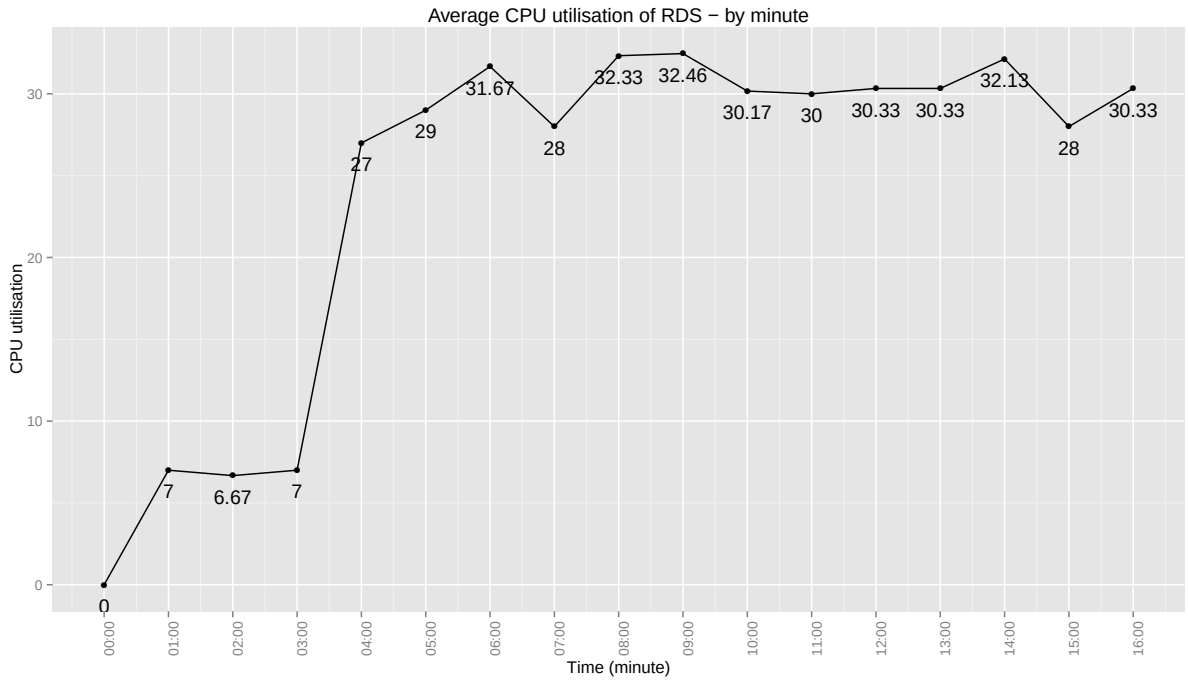
- vizualiziramo odzivne čase glede na operacijo (slika 7.7),
- izračunamo in vizualiziramo metriko *kapaciteta* (slika 7.8).

Rezultat obdelave in vizualizacije podatkov je 5 grafov, ki pomagajo pri interpretaciji merjenja ene konfiguracije sistema. Slike 7.4 - 7.8 prikazujejo vizualizacijo za meritev z eno podatkovno bazo tipa db.m3.medium, eno instanco tipa m3.medium za spletno aplikacijo in 2000 virtualnimi uporabniki.

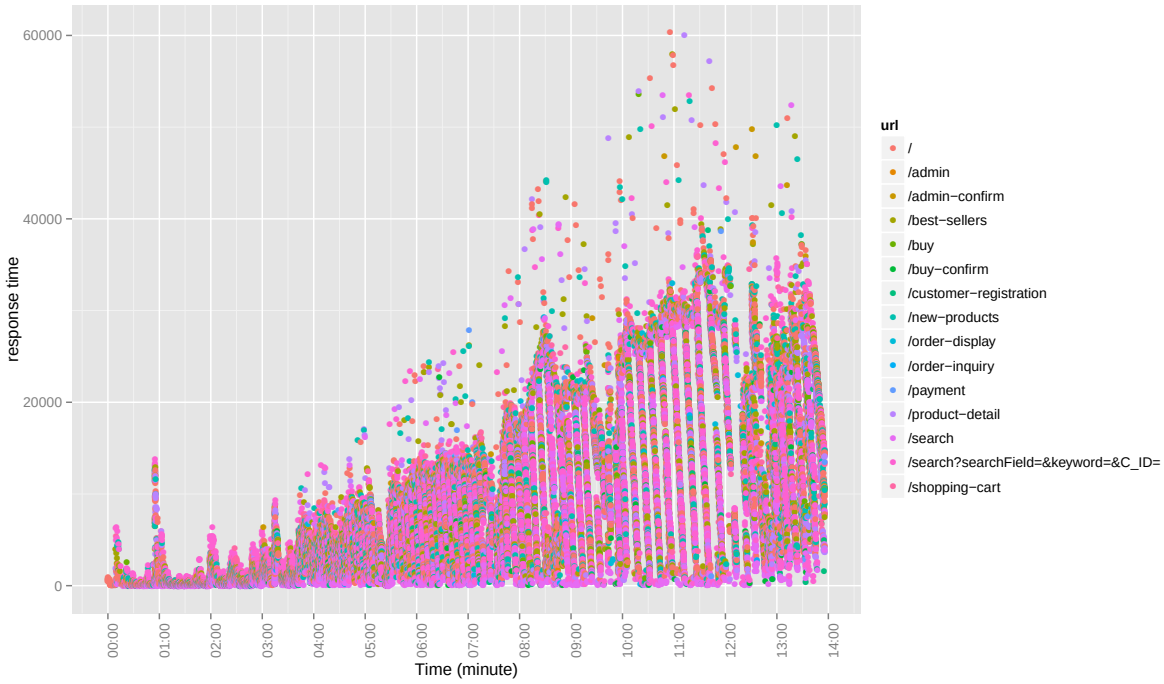
Na grafu 7.8, ki prikazuje kapaciteto, rdeča črta označuje prvo minuto, v kateri je bilo več kot 10 % SLO kršitev. To pomeni, da je med 5. in 6. minuto prišlo do SLO kršitev, zato kapaciteto izračunamo za obdobje med 4. in 5. minuto. Besedilo `req.=4809(4292)/VU=(563.0)` na grafu 7.8 pomeni, da bi se med 4. in 5. minuto teoretično moralo poslati 4809 zahtevkov, upoštevajoč 7 sekundni čas za razmislek, ki ga določa standard TPC-W, in



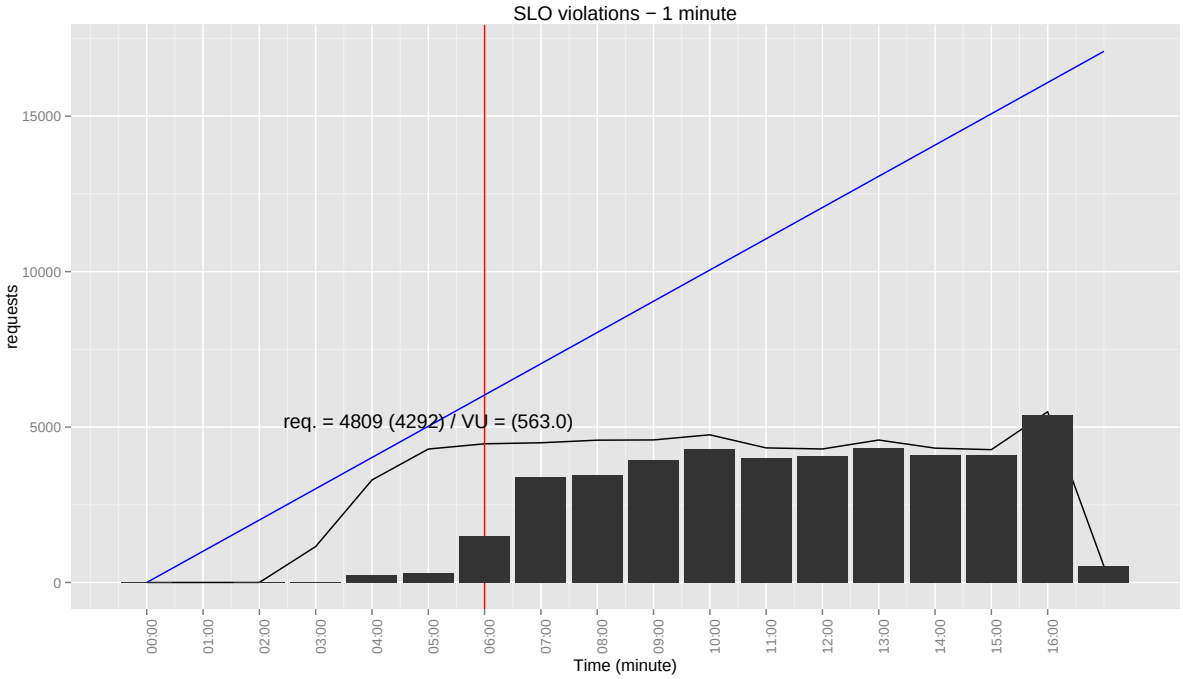
Slika 7.5: Povprečna izkoriščenost CPU za meritev z eno podatkovno bazo tipa db.m3.medium, eno instanco tipa m3.medium za spletno aplikacijo in 2000 virutalnimi uporabniki.



Slika 7.6: Izkoriščenost CPU podatkovne baze za meritev z eno podatkovno bazo tipa db.m3.medium, eno instanco tipa m3.medium za spletno aplikacijo in 2000 virtualnimi uporabniki.



Slika 7.7: Vizualizacija odzivnih časov glede na operacijo za meritev z eno podatkovno bazo tipa db.m3.medium, eno instanco tipa m3.medium za spletno aplikacijo in 2000 virtualnimi uporabniki.



Slika 7.8: Vizualizacija kapacitete za meritev z eno podatkovno bazo tipa db.m3.medium, eno instanco tipa m3.medium za spletno aplikacijo in 2000 virtualnimi uporabniki. Rdeča črta prikazuje prvo minuto, v kateri je bilo SLO kršitev večje od 10 %.

0 sekund odzivnega časa. Število 4292 označuje, da se je dejansko poslalo 4292 zahtevkov. Število 563.0 označuje kapaciteto med 4. in 5. minuto, ki se izračuna po naslednji enačbi:

$$\frac{\frac{2M-1}{2}}{S} * V \tag{7.1}$$

kjer je:

M - minuta, ko je še ni bilo kršitev SLO,

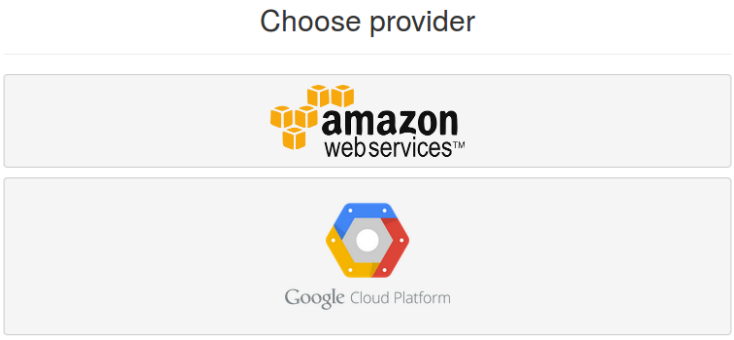
V - število simuliranih virtualnih uporabnikov,

S - dolžina scenarija v minutah.

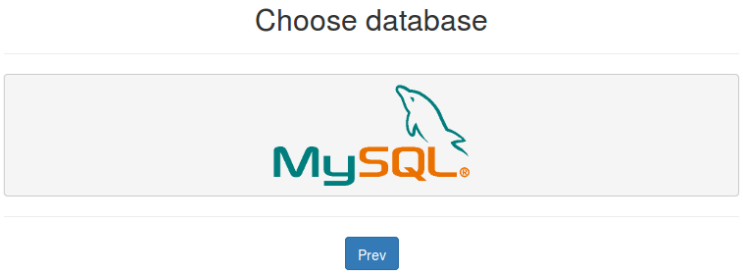
7.5 Spletna aplikacija

Spletno aplikacijo smo razvili zaradi uporabniku bolj prijazne uporabe razvitih orodij za nameščanje spletne trgovine v oblak in za nameščanje programa za obremenitev v oblak. Spletna aplikacija ponuja samo grafični vmesnik, ki omogoča enostavnejšo uporabo razvitih orodij, saj le-ti vsebujejo samo vmesnik za ukazno vrstico. Spletna aplikacija je razvita v programskem jeziku Python in uporablja ogrodje Django za grajenje spletnih strani. Uporabniški vmesnik je napisan v programskem jeziku JavaScript z uporabo programskega ogrodja AngularJS.

Spletna aplikacija vsebuje 6 korakov, s katerimi nastavimo in namestimo spletno aplikacijo in program za generiranje obremenitve v oblak. Na prvem koraku izberemo oblak, na katerem želimo izvesti test (slika 7.9). Na drugem koraku izberemo podatkovno bazo, na katero želimo namestiti podatkovno bazo (slika 7.10). Trenutno je na voljo samo podatkovna baza MySQL. Na tretjem in četrtem koraku vnesemo podatke za povezovanje z izbranim oblakom in nastavimo parametre za nameščanje spletne trgovine ter podatkovne baze v izbran oblak (slika 7.11). Na petem koraku vnesemo podatke za obremenitveni test in naložimo scenarij, ki ga želimo izvesti (slika 7.12). Zadnji, šesti korak, prikazuje potek procesa nameščanja spletne trgovine in testiranja (slika 7.13). Po uspešno končanem testu si uporabnik lahko

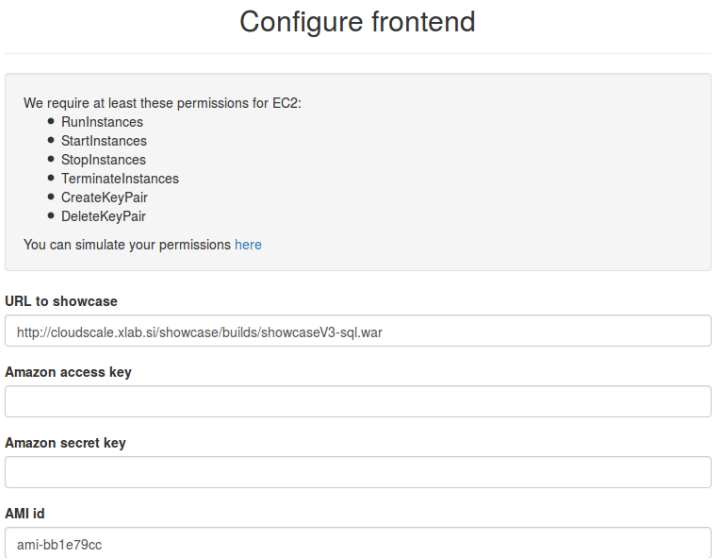


Slika 7.9: Prvi korak pri izvajanju meritve s spletno aplikacijo. Uporabnik izbere oblak, na katerem želi izvesti test.

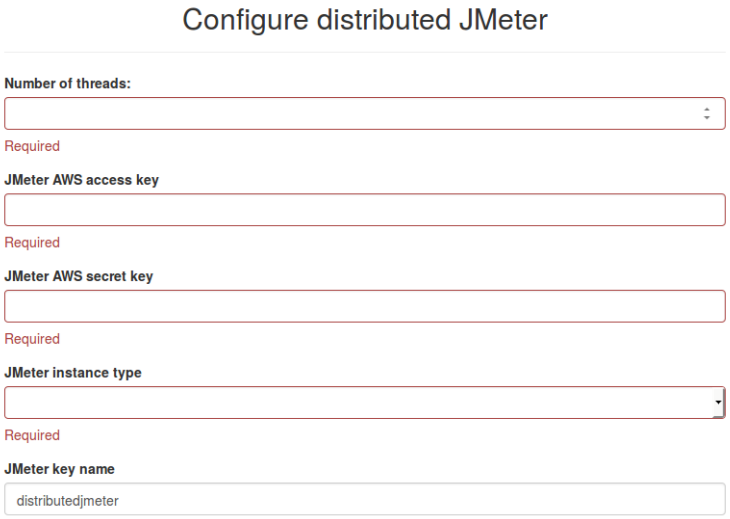


Slika 7.10: Drugi korak pri izvajanju meritve s spletno aplikacijo. Uporabnik izbere podatkovno bazo, na kateri želi izvesti test.

prenese dnevnik zapisov o programu za obremenitveno testiranje, datoteko `response-times-over-time.csv`, ki vsebuje zapise o poslanih zahtevah in odzivnih časih, in grafe, ki jih generira program za obdelavo in vizualizacijo podatkov.



Slika 7.11: Tretji korak pri izvajanju meritve s spletno aplikacijo. Uporabnik vnese podatke za povezovanje z izbranim oblakom in parametre za nameščanje spletne trgovine.



Slika 7.12: Peti korak pri izvajanju meritve s spletno aplikacijo. Uporabnik vnese podatke za nameščanje podatkovne baze v izbran oblak.

Slika 7.13: Šesti korak pri izvajanju meritve s spletno aplikacijo. Izpis napredka o nameščanju in testiranju spletne aplikacije v izbranem oblaku. Po končani meritvi, si uporabnik lahko prenese dnevnik zapisov o programu za obremenitveno testiranje, datoteko `response-times-over-time.csv` in grafe.

Poglavje 8

Meritve in rezultati

V tem poglavju predstavimo metodologijo testiranja, meritve in rezultate testiranja ovrednotene z metrikami, ki smo jih definirali v sekciji 6.

8.0.1 Metodologija

Metodologijo, ki jo uporabljamo za izvajanje testov lahko povzamemo v 4 korakih:

- izbira konfiguracije,
- testiranje,
- zbiranje in obdelava rezultatov,
- interpretacija rezultatov.

Prvi korak je **izbira konfiguracije** (5.1), ki jo nudi izbran oblak. Konfiguracije izbiramo, tako da primerjamo tipe instanc obeh oblakov in testiramo tiste, ki so si po zmogljivosti najbolj podobne. Ker storitev Cloud SQL ne določa zmogljivosti CPU za podatkovno bazo, tipe instanc za podatkovno bazo primerjamo samo po zmogljivosti RAM. Tabeli 8.1 in 8.2 prikazujeta podobne tipe instanc za storitvi CloudSQL in RDS ter podobne tipe instanc za storitvi Compute Engine in EC2.

RDS	Max. conn.	Cloud SQL	Max. conn.
db.m3.medium - 3.75 GB RAM	296	D8 - 4 GB RAM	1000
db.m3.large - 7.5 GB RAM	604	D16 - 8 GB RAM	2000
db.m3.xlarge - 15 GB RAM	1232	D32 - 16 GB RAM	4000

Tabela 8.1: Primerljivi tipi instanc za storitvi RDS in Cloud SQL za relacijske podatkovne baze.

EC2	vCPU	RAM	Compute engine	vCPU	RAM
m3.medium	1	3.75	n1-standard-1	1	3.75
m3.large	2	7.5	n1-standard-2	2	7.5
m3.xlarge	4	15	n1-standard-4	4	15
m3.2xlarge	8	30	n1-standard-8	8	30

Tabela 8.2: Primerjava tipov instanc storitev EC2 in Compute Engine za oskrbovanje z virtualnimi stroji.

Pri izbiri tipa instanc se osredotočimo na standardne tipe instanc, ki ponujajo najboljše razmerje računalniških virov. Izognemo se tipu instanc, ki optimizirajo določen vir - npr. CPU ali RAM, ter tipu instanc, ki ne ponujajo konstantne zmogljivosti, saj so testi v tem primeru težko ponovljivi in zato težko primerljivi. Pri oblaku GCP sta tipa instanc z nekonstantno zmogljivostjo *f1-micro* in *g1-small*, pri oblaku AWS pa je to družina instanc tipa *t2*.

Drugi korak pri določanju metodologije je izvedba testiranja. Testiranje izvedemo z 2000, 4000, 6000 ali 8000 virtualnimi uporabniki, ker en primerek programa JMeter lahko simulira 2000 virtualnih uporabnikov. Pred vsakim testiranjem na novo namestimo celotno konfiguracijo, saj s tem zagotovimo, da so vse meritve zagnane iz istega začetnega stanja. S tem tudi preprečimo, da bi katerikoli strežnik shranjeval podatke v predpomnilnik (angl. *cache*).

Najprej izvedemo meritve za ročno skaliranje in določimo število virtualnih uporabnikov, s katerimi obremenitveno testiramo spletno aplikacijo. Število virtualnih uporabnikov določimo v dveh korakih. Najprej zaženemo

test s številom virtualnih uporabnikov, za katerega menimo, da jih bo ena konfiguracija še zmožna obdelati. Nato testiranje zaženemo z večjim oz. manjšim številom virtualnih uporabnikov, odvisno od rezultatov prejšnega testiranja. Primer: s prvim testiranjem s 4000 virtualnimi uporabniki smo izmerili kapaciteto 1250 virtualnih uporabnikov za določeno konfiguracijo sistema, zato v naslednjem testiranju test zaženemo z 2000 virtualnimi uporabniki za isto konfiguracijo. Na ta način zmanjšamo stopnjo povečevanja virtualnih uporabnikov na minuto in tako dobimo bolj natančno kapaciteto kot s prejšnim testiranjem, kjer je nevarnost, da virtualni stroj že na začetku preobremenimo s prevelikim številom virtualnih uporabnikov zaradi prevelike stopnje povečevanja virtualnih uporabnikov na minuto.

Dobljeno število virtualnih uporabnikov nato uporabimo še pri ostalih meritvah za ročno skaliranje, z istim številom virtualnih strojev za izvajanje spletne trgovine in tremi različnimi tipi instanc za podatkovno bazo. Na oblaku AWS smo za podatkovno bazo uporabili tipe instanc **db.m3.medium**, **db.m3.large** in **db.m3.xlarge**. Na oblaku GCP smo za podatkovno bazo uporabili tipe instanc **D8**, **D16** in **D32**.

Po tem, ko smo določili število virtualnih uporabnikov za posamezno meritev, smo enake meritve uporabili tudi za avtomatsko skaliranje. Pri avtomatskem skaliranju smo vsako konfiguracijo sistema testirali trikrat zaporedoma in izračunali povprečno kapaciteto vseh treh meritev.

Tretji korak pri metodologiji je zbiranje rezultatov. Po končanem testu z virtualnega stroja, kjer se je izvajal program za obremenitveno testiranje, zberemo podatke o testu jih obdelamo, izdelamo grafe in kalkulacije ter ponudimo uporabniku za interpretacijo.

Četrti korak pri metodologiji je interpretacija rezultatov. Interpretacijo rezultatov moramo izvesti ročno, saj je potrebno pregledati grafe in interpretirati dobljene številke.

8.1 Izvedba meritev

Meritve smo izvedli po zgornji metodologiji z uporabo orodij in aplikacij, ki so opisani v razdelku 7. Med izvajanjem meritev smo naleteli na nekaj težav s programsko opremo, ki jo uporabljamo za izvajanje spletne trgovine in so se pokazale šele ob povečani obremenitvi na spletno aplikacijo ter ob normalni uporabi niso vidne. Težave smo rešili tako, da smo optimizirali nastavitve programske opreme in operacijskega sistema, ki ga uporabljamo za izvajanje spletne trgovine.

Meritve smo najprej izvedli na oblaku GCP, nato pa po preslikavi po tabelah 8.2 in 8.1 še enake meritve na oblaku AWS, zato da dobimo primerljive rezultate. Meritve smo izvedli samo za tipe instanc *m3.medium, n1-standard-1* in *m3.large, n1-standard-2* za spletno aplikacijo v kombinaciji z vsemi tremi tipi instanc za podatkovno bazo. Meritev za ostale tipe instanc za spletno trgovino nismo izvedli zaradi visokih stroškov testiranja. Na obeh oblakih smo presegli omejitev 100$. V stroške so vključene tudi meritve, ki smo jih opravili za razvijanje aplikacij.

Metriko *kapaciteta* smo najprej izmerili z ročnim skaliranjem, da smo dobili občutek, kakšne meritve moramo izvajati z vklopljenim avtomatskim skaliranjem. Po tem ko smo izmerili kapaciteto za ročno skaliranje, smo ponovili še iste meritve z vklopljenim avtomatskim skaliranjem. Vsako meritev z vklopljenim avtomatskim skaliranjem smo izvedli 3-krat zapored. Končna kapaciteta je povprečna kapaciteta treh zaporednih meritev. Tak postopek smo ubrali, ker se je zgodilo, da je bila kapaciteta za vsako meritev z avtomatskim skaliranjem drugačna.

Meritve in njihove rezultate ovrednotene z metrikama *kapaciteta* in *skalabilnost* opišemo v razdelku 8.2.

8.2 Rezultati meritev

V tem razdelku opišemo rezultate meritev ovrednotenih z metrikama *kapaciteta* in *skalabilnost*, ki smo ju definirali v razdelku 6. Vse skupaj smo za vsak

oblak opravili preko 50 meritev. Nekaj meritev smo opravili samo zato, da vidimo, kaj se zgodi z metriko, če spremenimo samo en parameter v nastavitvah - npr. velikost bazena povezav do podatkovne baze, zato v tem razdelku predstavimo in opišemo samo najbolj pomembne meritve. Največ smo spreminjali parameter za nastavitev velikosti bazena povezav do podatkovne baze in parameter za število virtualnih uporabnikov.

8.2.1 Rezultati za oblak GCP

Rezultati za ročno skaliranje

V tabeli 8.3 so zbrani rezultati za meritve z ročnim skaliranjem za tipe instanc `n1-standard-1` in `n1-standard-2` za spletno aplikacijo in tipe instanc D8, D16 in D32 za podatkovno bazo.

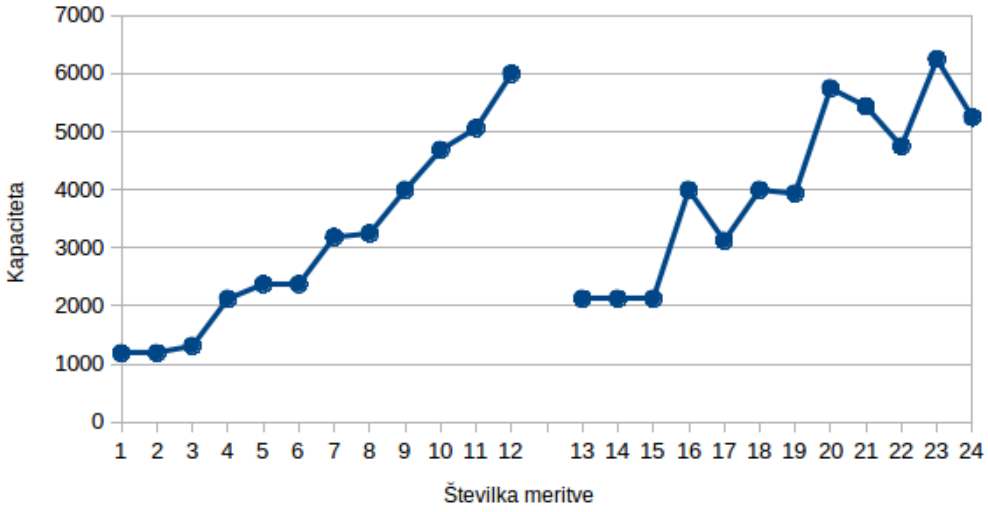
Vizualizacija meritev iz tabele 8.3 z metriko **skalabilnost** pokaže, da kapaciteta za tip instance `n1-standard-1` za izvajanje spletne aplikacije z ročnim dodajanjem virov in povečevanjem zmogljivosti podatkovne baze kapaciteta linearno narašča. Pri instancah tipa `n1-standard-2` pri nekaterih meritvah (17, 22, 24) opazimo, da je kapaciteta kljub povečanju zmogljivosti podatkovne baze bila manjša kot pri meritvah z manj zmogljivo podatkovno bazo. Če za omenjene tri meritve pogledamo graf za izkoriščenost CPU, pri nobeni meritvi ni bila nad 75 %. S tem smo izločili možnost, da je ozko grlo CPU na virtualnih strojih za izvajanje spletne aplikacije. Ostaneta še dve možnosti za ozko grlo, in sicer še izkoriščenost CPU podatkovne baze ali velikost bazena povezav do podatkovne baze. Bazen je pri vseh treh meritvah do konca izčrpan. Informacije o izkoriščenosti CPU za podatkovno bazo na oblaku GCP ni možno pridobiti.

Rezultati za avtomatsko skaliranje

V tabeli 8.4 so zbrani rezultati enakih meritev, kot smo jih opravili z ročnim skaliranjem, le da je bilo tokrat skaliranje avtomatsko. Kot smo že omenili, smo vsako meritev ponovili 3-krat in zabeležili kapaciteto. Končna kapaciteta

#	Tip PB	Tip apl.	Št. PB	Št. apl.	VU	Vel. bazena	Kap.
1	D8	n1-standard-1	1	1	2000	1000	1188
2	D16	n1-standard-1	1	1	2000	2000	1188
3	D32	n1-standard-1	1	1	2000	4000	1313
4	D8	n1-standard-1	1	2	4000	500	2125
5	D16	n1-standard-1	1	2	4000	1000	2375
6	D32	n1-standard-1	1	2	4000	2000	2375
7	D8	n1-standard-1	1	3	4000	333	3188
8	D16	n1-standard-1	1	3	4000	666	3250
9	D32	n1-standard-1	1	3	4000	1333	4000
10	D8	n1-standard-1	1	4	6000	250	4688
11	D16	n1-standard-1	1	4	6000	500	5063
12	D32	n1-standard-1	1	4	6000	1000	6000
13	D8	n1-standard-2	1	1	4000	1000	2125
14	D16	n1-standard-2	1	1	4000	2000	2125
15	D32	n1-standard-2	1	1	4000	4000	2125
16	D8	n1-standard-2	1	2	4000	500	4000
17	D16	n1-standard-2	1	2	4000	1000	3125
18	D32	n1-standard-2	1	2	4000	2000	4000
19	D8	n1-standard-2	1	3	6000	333	3938
20	D16	n1-standard-2	1	3	6000	666	5750
21	D32	n1-standard-2	1	3	6000	1333	5438
22	D8	n1-standard-2	1	4	8000	250	4750
23	D16	n1-standard-2	1	4	8000	500	6250
24	D32	n1-standard-2	1	4	8000	1000	5750

Tabela 8.3: Tabela meritev za oblak GCP z ročnim skaliranjem za tipa instanc `n1-standard-1` in `n1-standard-2` za spletno aplikacijo in tipe instanc D8, D16, D32 za podatkovno bazo.



Slika 8.1: Vizualizacija vseh meritev iz tabele 8.3 z metriko **skalabilnost** za ročno skaliranje.

v tabeli 8.4 je povprečje vseh treh kapacitet. Primer: za meritev s številko 1 iz tabele 8.4 smo dobili tri različne kapacitete: 688, 438 in 188. Končna kapaciteta je v tem primeru:

$$\frac{688 + 438 + 188}{3} = 438$$

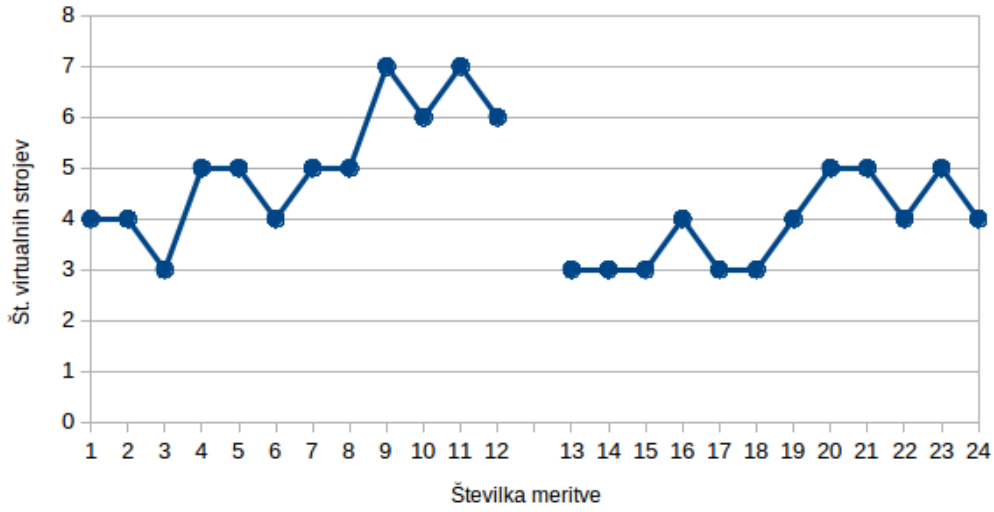
Pri avtomatskem skaliranju smo beležili tudi število instanc za spletno aplikacijo, do katerega je oblak skaliral, da je lahko podprl željeno obremenitev. Slika 8.2 prikazuje kako število virtualnih strojev za izvajanje spletne aplikacije narašča z večanjem obremenitve.

Vizualizacija **skalabilnosti** z grafom pokaže (slika 8.3), da s povečevanjem velikosti bazena povezav do podatkovne baze ne pridobimo na kapaciteti, ampak je celo manjša. Našo tezo potrjujejo tudi meritve za tip instance `n1-standard-2` za izvajanje spletne aplikacije, kjer lahko opazimo enak vzorec.

Iz te ugotovitve lahko sklepamo, da je za avtomatsko skaliranje najboljše

#	Tip PB	Tip apl.	Št. PB	**Št. apl.**	VU	Vel. bazena	**Kap.**
1	D8	n1-standard-1	1	3	2000	1000	438
2	D16	n1-standard-1	1	3	2000	2000	980
3	D32	n1-standard-1	1	4	2000	4000	1021
4	D8	n1-standard-1	1	5	4000	500	875
5	D16	n1-standard-1	1	5	4000	1000	375
6	D32	n1-standard-1	1	4	4000	2000	542
7	D8	n1-standard-1	1	5	4000	333	625
8	D16	n1-standard-1	1	5	4000	666	792
9	D32	n1-standard-1	1	7	4000	1333	563
10	D8	n1-standard-1	1	6	6000	250	688
11	D16	n1-standard-1	1	7	6000	500	688
12	D32	n1-standard-1	1	6	6000	1000	750
13	D8	n1-standard-2	1	3	4000	1000	1625
14	D16	n1-standard-2	1	3	4000	2000	1375
15	D32	n1-standard-2	1	3	4000	4000	1625
16	D8	n1-standard-2	1	4	4000	450	1625
17	D16	n1-standard-2	1	3	4000	1000	1625
18	D32	n1-standard-2	1	4	4000	2000	1042
19	D8	n1-standard-2	1	4	6000	333	1313
20	D16	n1-standard-2	1	5	6000	666	1813
21	D32	n1-standard-2	1	5	6000	1333	1438
22	D8	n1-standard-2	1	4	6000	250	1083
23	D16	n1-standard-2	1	5	8000	500	1417
24	D32	n1-standard-2	1	4	8000	1000	1250

Tabela 8.4: Tabela meritev za oblak GCP z avtomatskim skaliranjem za tipe instanc `n1-standard-1` in `n1-standard-2` za spletno aplikacijo in tipe instanc D8, D16, D32 za podatkovno bazo.



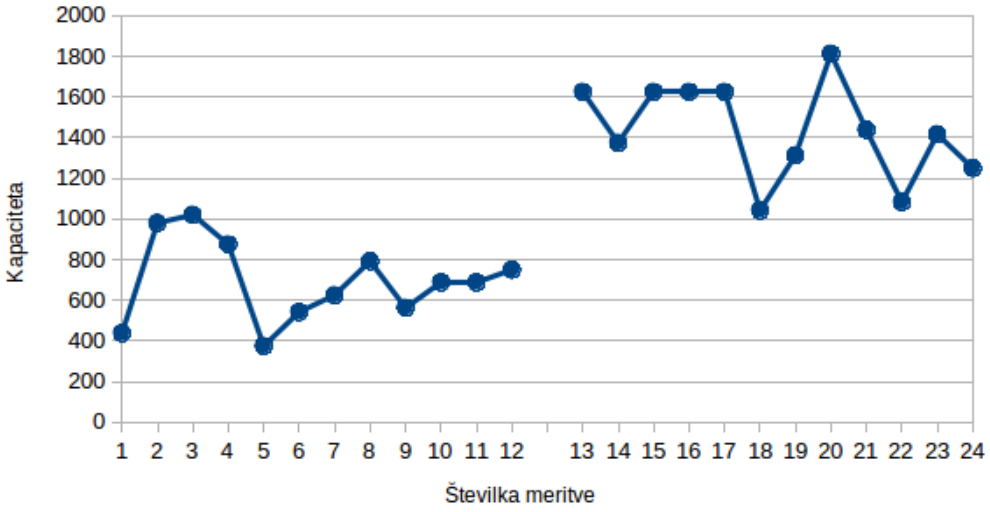
Slika 8.2: Graf, ki za vsako meritev iz tabele 8.4 za oblak GCP prikazuje število virtualnih strojev, ki jih je storitev za avtomatsko skaliranje zagnala, da je podprla simulirano obremenitev.

nastaviti velikost bazena povezav do podatkovne baze kar na maksimalno število sočasnih povezav do podatkovne baze.

Iz grafa je tudi razvidno, da ko povečamo zmogljivost virtualnih strojev za izvajanje spletne aplikacije, kapaciteta za meritve s tipom instanc `n1-standard-2` za izvajanje spletne trgovine močno naraste. Opazimo tudi, da za meritve z avtomatskim skaliranjem oblak GCP potrebuje veliko več instanc za izvajanje spletne trgovine, kot z ročnim skaliranjem, pri tem pa celo omogoča manjšo kapaciteto. Razlog za to je lahko prehitra stopnja povečevanja virtualnih uporabnikov na minuto.

8.2.2 Rezultati za oblak AWS

Na oblaku AWS smo izvedli enake meritve kot na oblaku GCP, le da smo tipe instanc za podatkovno bazo preslikali po tabeli 8.1 in tipe instanc za izvajanje spletne trgovine po tabeli 8.2. Parameter za velikost bazena pove-



Slika 8.3: Vizualizacija kapacitete za meritve na oblaku GCP iz tabele 8.4 z vklopljenim avtomatskim skaliranjem.

zav do podatkovne baze smo prilagodili omejitvam storitve RDS za relacijske podatkovne baze. Razlika v omenjenem parametru za podobne tipe instanc je zbrana v tabeli 8.1.

Prav tako, kot na oblaku GCP smo tudi na oblaku AWS najprej meritve izvedli z ročnim skaliranjem in nato še z vklopljenim avtomatskim skaliranjem.

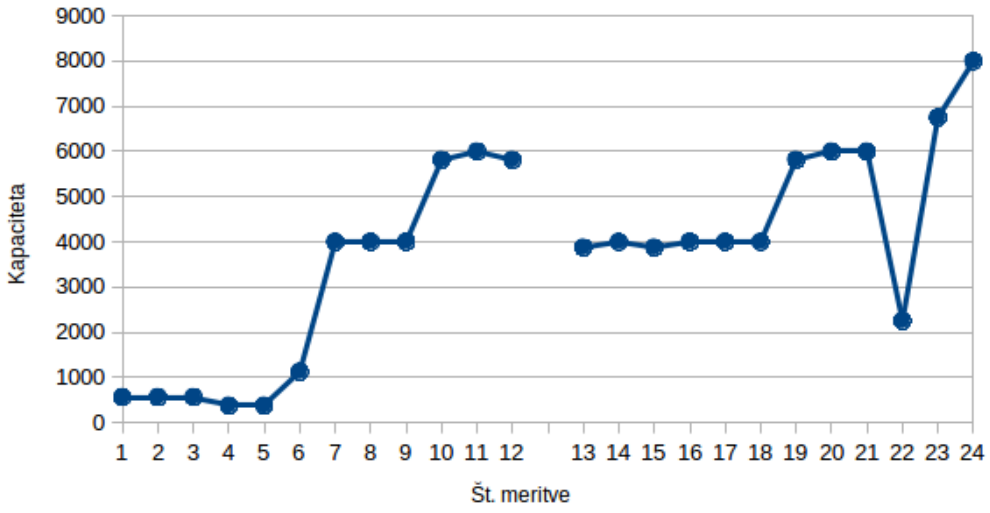
Rezultati za ročno skaliranje

V tabeli 8.5 so zbrane meritve z ročnim skaliranjem števila virtualnih strojev za izvajanje spletne trgovine in izračunana kapaciteta za posamezno meritev.

Vizualizacija skalabilnosti z grafom pokaže (slika 8.4), da kapaciteta z ročnim povečevanjem zmogljivosti podatkovne baze in števila virtualnih strojev za izvajanje spletne aplikacije narašča, razen pri meritvi 22, kjer je kapaciteta 2250. Podrobnejša analiza pokaže, da je ozko grlo v tem primeru CPU na podatkovni bazi. Razlog za povečan CPU pa je prehitra stopnja

#	Tip PB	Tip apl.	Št. PB	Št. apl.	Vel. bazena	Št. VU	**Kap.**
1	db.m3.medium	m3.medium	1	1	290	2000	563
2	db.m3.large	m3.medium	1	1	600	2000	563
3	db.m3.xlarge	m3.medium	1	1	1230	2000	563
4	db.m3.medium	m3.medium	1	2	145	4000	375
5	db.m3.large	m3.medium	1	2	300	4000	375
6	db.m3.xlarge	m3.medium	1	2	615	4000	1125
7	db.m3.medium	m3.medium	1	3	97	4000	4000
8	db.m3.large	m3.medium	1	3	200	4000	4000
9	db.m3.xlarge	m3.medium	1	3	410	4000	4000
10	db.m3.medium	m3.medium	1	4	73	6000	5813
11	db.m3.large	m3.medium	1	4	150	6000	6000
12	db.m3.xlarge	m3.medium	1	4	308	6000	5813
13	db.m3.medium	m3.large	1	1	290	2000	3875
14	db.m3.large	m3.large	1	1	600	2000	4000
15	db.m3.xlarge	m3.large	1	1	1230	2000	3875
16	db.m3.medium	m3.large	1	2	145	4000	4000
17	db.m3.large	m3.large	1	2	300	4000	4000
18	db.m3.xlarge	m3.large	1	2	615	4000	4000
19	db.m3.medium	m3.large	1	3	97	4000	5813
20	db.m3.large	m3.large	1	3	200	4000	6000
21	db.m3.xlarge	m3.large	1	3	410	4000	6000
22	db.m3.medium	m3.large	1	4	73	8000	2250
23	db.m3.large	m3.large	1	4	150	8000	6750
24	db.m3.xlarge	m3.large	1	4	308	8000	8000

Tabela 8.5: Tabela meritev za oblak AWS z ročnim skaliranjem za tipe instanc m3.medium in m3.large za spletno aplikacijo z tipi instanc db.m3.medium, db.m3.large in db.m3.xlarge za podatkovno bazo



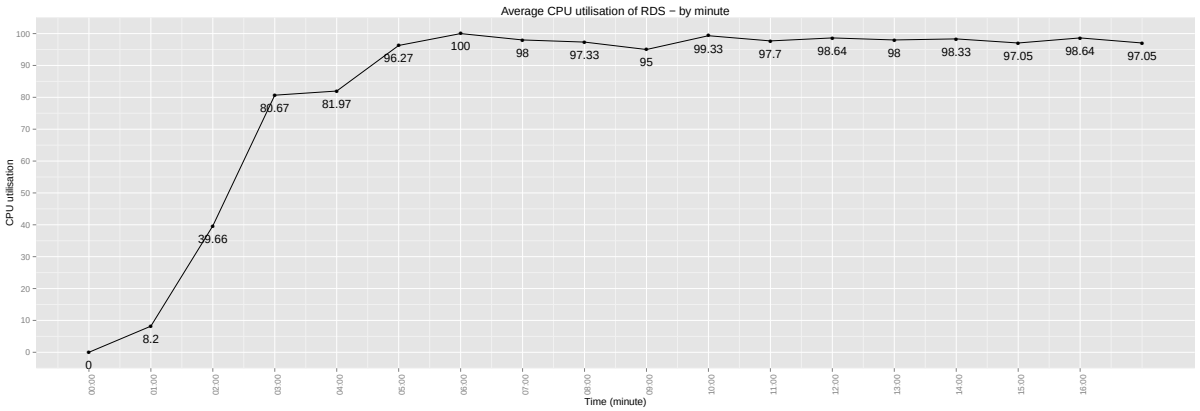
Slika 8.4: Vizualizacija kapacitete meritev iz tabele 8.5 za oblak AWS z ročnim skaliranjem.

povečevanja virtualnih uporabnikov na minuto in že po 5-ih minutah preobremenimo podatkovno bazo (slika 8.5), kar povzroči daljše odzivne čase in posledično manjšo kapaciteto.

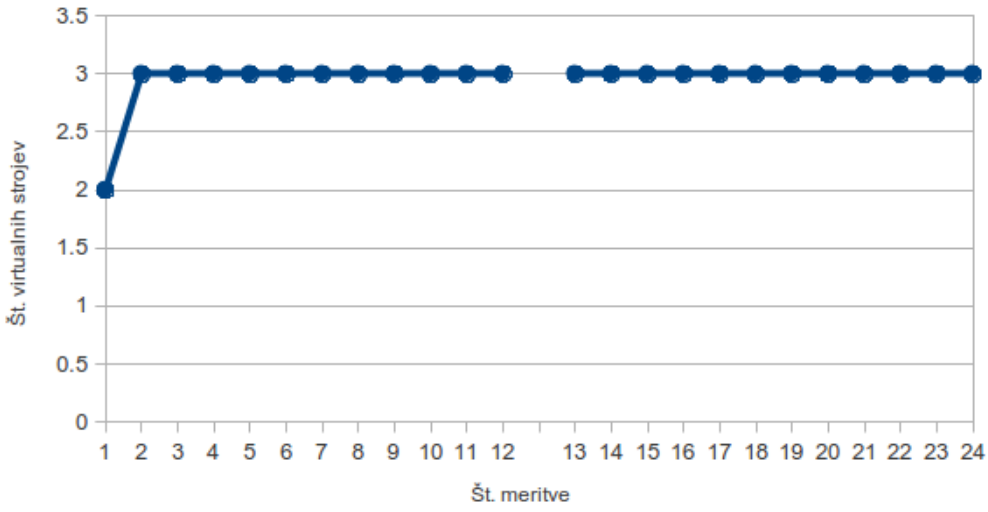
Rezultati za avtomatsko skaliranje

V tabeli 8.6 so zbrane meritve z avtomatskim skaliranjem in njihove kapacitete. Pri izvajanju meritev z avtomatskim skaliranjem smo poleg kapacitete beležili še število virtualnih strojev, ki jih je storitev EC2 zagnala, zato da je podprla simulirano obremenitev. Graf na sliki 8.6 prikazuje, da je storitev avtomatskega skaliranja v oblaku AWS zagnala največ 3 virtualne stroje med eno meritvijo.

Vizualizacija **skalabilnosti** z grafom (slika 8.7) pokaže, da kapaciteta z večanjem velikosti bazena povezav do podatkovne baze zelo počasi narašča. V primeru meritev 19-24 je ta celo slabša kot pri meritvah 13-17, ki se giblje okrog 2000 virtualnih uporabnikov. Tudi pri oblaku AWS z avtomatskim



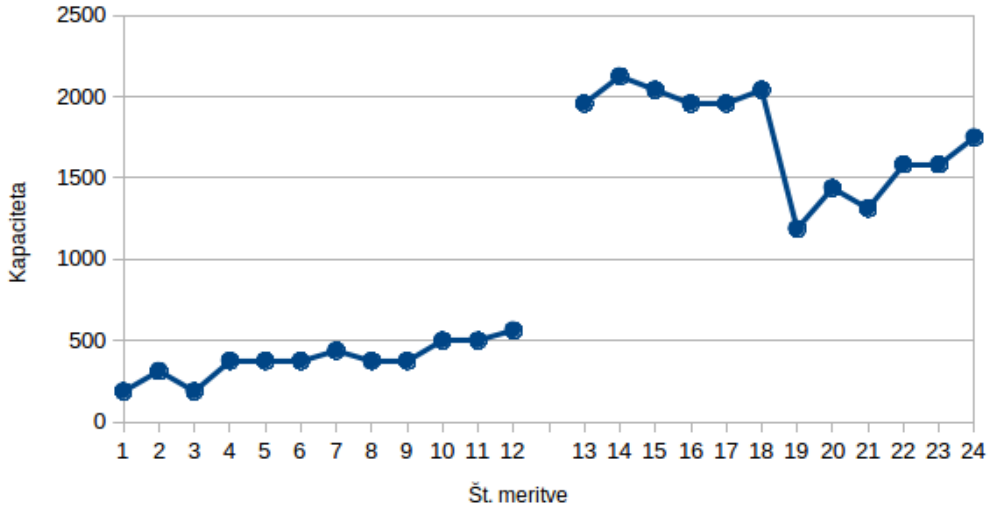
Slika 8.5: Izkoriščenost CPU za meritev s številko 22 na oblaku AWS.



Slika 8.6: Graf, ki prikazuje število virtualnih strojev, ki jih je storitev avtomatskega skaliranja na oblaku AWS zagnala, da je podprla željeno obremenitev.

#	Tip PB	Tip apl.	Št. PB	**Št. apl.**	Vel. bazena	St. VU	**Kap.**
1	db.m3.medium	m3.medium	1	2	290	2000	188
2	db.m3.large	m3.medium	1	3	600	2000	188
3	db.m3.xlarge	m3.medium	1	3	1230	2000	188
4	db.m3.medium	m3.medium	1	3	145	4000	375
5	db.m3.large	m3.medium	1	3	300	4000	375
6	db.m3.xlarge	m3.medium	1	3	615	4000	375
7	db.m3.medium	m3.medium	1	3	97	4000	438
8	db.m3.large	m3.medium	1	3	200	4000	375
9	db.m3.xlarge	m3.medium	1	3	410	4000	375
10	db.m3.medium	m3.medium	1	3	73	6000	500
11	db.m3.large	m3.medium	1	3	150	6000	500
12	db.m3.xlarge	m3.medium	1	3	308	6000	563
13	db.m3.medium	m3.large	1	3	290	2000	1958
14	db.m3.large	m3.large	1	3	600	2000	2125
15	db.m3.xlarge	m3.large	1	3	1230	2000	2042
16	db.m3.medium	m3.large	1	3	145	4000	1958
17	db.m3.large	m3.large	1	3	300	4000	1958
18	db.m3.xlarge	m3.large	1	3	615	4000	3458
19	db.m3.medium	m3.large	1	3	97	4000	1188
20	db.m3.large	m3.large	1	3	200	4000	1438
21	db.m3.xlarge	m3.large	1	3	410	4000	1313
22	db.m3.medium	m3.large	1	3	73	6000	1583
23	db.m3.large	m3.large	1	3	150	6000	1583
24	db.m3.xlarge	m3.large	1	3	308	6000	1750

Tabela 8.6: Tabela meritev za oblak AWS z avtomatskim skaliranjem za tipe instanc m3.medium in m3.large za spletno aplikacijo z tipi instanc db.m3.medium, db.m3.large in db.m3.xlarge za podatkovno bazo.



Slika 8.7: Vizualizacija skalabilnosti za meritve na oblaku AWS z vklopljenim avtomatskim skaliranjem.

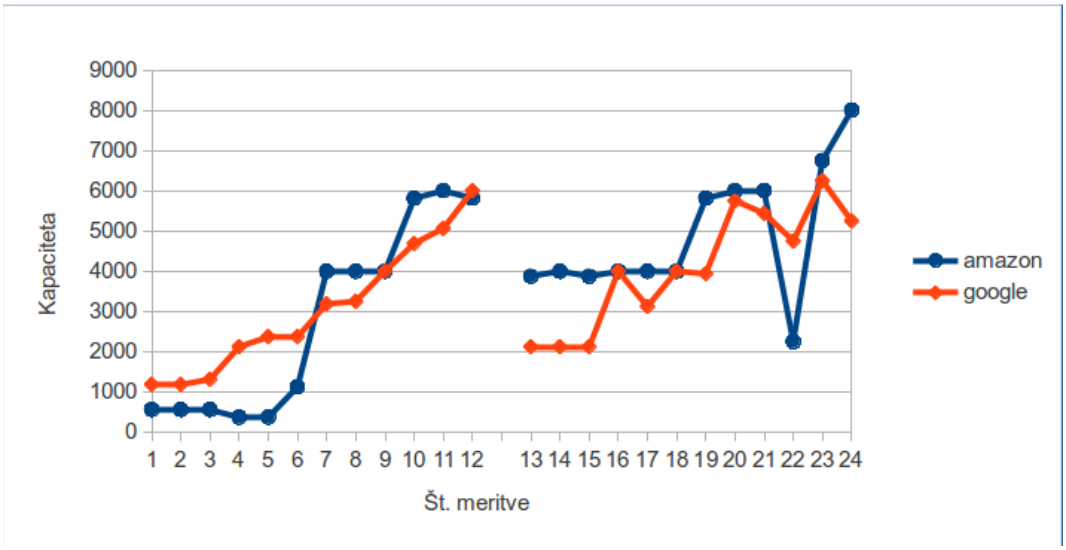
skaliranjem potrebujemo več virtualnih strojev za podporo manjši kapaciteti kot z ročnim skaliranjem.

8.3 Primerjava rezultatov meritev

8.3.1 Primerjava rezultatov ročnega skaliranja

Vizualizacija rezultatov meritev iz razdelka 8.2 na isti graf, da bolj jasno sliko o razlikah med oblakoma. Vizualizacija **skalabilnosti** na isti graf (slika 8.8) pokaže, da je kapaciteta na oblaku GCP za prvih nekaj meritev (1-6) boljša kot pri istih meritvah na oblaku AWS. Z dodajanjem virtualnih strojev za izvajanje spletne aplikacije in večanjem zmogljivosti podatkovne baze opazimo, da se kapaciteta na oblaku AWS močno poveča (meritve 7-12), medtem ko na oblaku GCP z dodajanjem virov in večanjem zmogljivosti virov opazimo počasnejšo rast kapacitete.

Povečanje zmogljivosti virtualnih strojev za izvajanje spletne aplikacije



Slika 8.8: Vizualizacija **skalabilnosti** na isti graf za oba oblaka za meritve z ročnim skaliranjem.

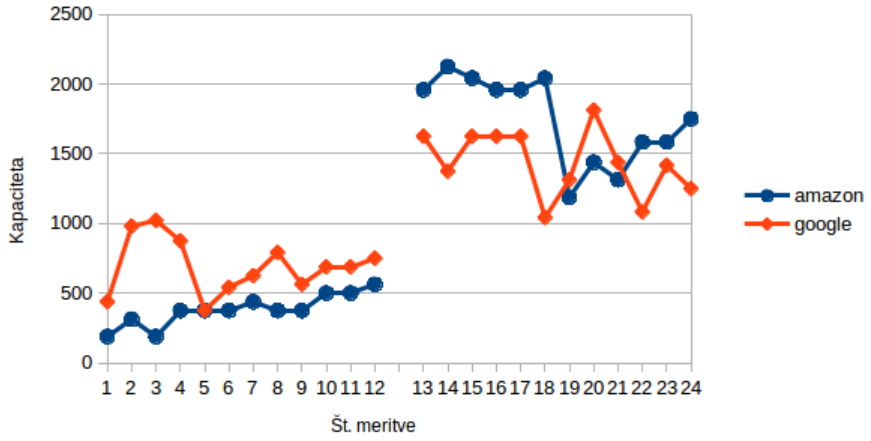
(meritve 13-24) na oblaku AWS bolj pripomore k povečanju kapacitete kot na oblaku GCP.

8.3.2 Primerjava rezultatov avtomatskega skaliranja

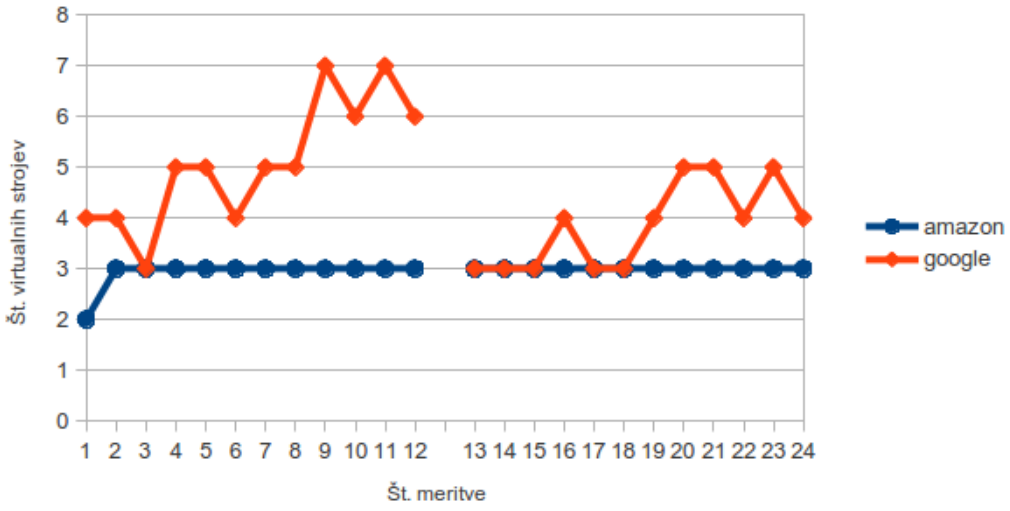
Vizualizacija **skalabilnosti** za oba oblaka z vklopljenim avtomatskim skaliranjem na isti graf (slika 8.9) pokaže, da je kapaciteta na oblaku GCP v splošnem večja kot na oblaku AWS za meritve 1-12. Kapaciteta za meritve 13-24, kjer smo povečali zmogljivost virtualnih strojev za izvajanje spletne trgovine, je ta v splošnem večja na oblaku AWS kot na oblaku GCP.

Vizualizacija števila virtualnih strojev (slika 8.10), ki jih posamezen oblak zažene, da podpre določeno obremenitev pokaže, da oblak GCP zažene več virtualnih strojev kot oblak AWS. To se zgodi zaradi različnih politik skaliranja.

Pri izvajanju meritev smo ugotovili, da pri ročnem skaliranju na končno kapaciteto veliko vpliva nastavitev velikosti bazena povezav do podatkovne baze in omejitev maksimalnega števila sočasnih povezavo do podatkovne



Slika 8.9: Vizualizacija **skalabilnosti** na isti graf za oba oblaka z vklopljenim avtomatskim skaliranjem.



Slika 8.10: Vizualizacija števila virtualnih strojev, ki jih posamezen oblak zažene, da podpre določeno obremenitev.

baze. Omenjeni nastavitvi sta zelo povezani. Opazili smo, da ko se bazen povezav do podatkovne baze izčrpa, na virtualnih strojih za izvajanje spletne trgovine izkoriščenost CPU močno naraste. To se zgodi zato, ker veliko povezav na aplikacijskem strežniku Tomcat čaka na obdelavo, ki pa ni možna zaradi izčrpanosti. Omenjena težava se pozna tudi na daljših odzivnih časih.

Pri testiranju meritev z avtomatskim skaliranjem smo ugotovili, da bi lahko uporabili še daljši scenarij od 16 minut, vendar bi dobili tudi drugačne rezultate. Problem, ki smo ga opazili med testiranjem meritev je, da so pri meritvah z ročnim skaliranjem, z več kot enim virtualnim strojem za izvajanje spletne trgovine, vsi virtualni stroji na voljo za sprejemanje zahtevkov takoj na začetku meritve, medtem ko je pri meritvah z avtomatskim skaliranjem na začetku scenarija na voljo samo en virtualni stroj za sprejemanje zahtevkov. Tako pri večjih obremenitvah s 4000 virtualnimi uporabniki in 6000 virtualnimi uporabniki pri avtomatskem skaliranju že na začetku preobremenimo virtualni stroj z zahtevki, kar povzroči zelo veliko izkoriščenost CPU, daljše odzivne čase in posledično kršitve SLO že na začetku izvajanja scenarija. Zamašitev sicer sproži operacijo skaliranja, vendar ta traja predolgo, da ne bi prišlo do kršitev SLO.

Poglavje 9

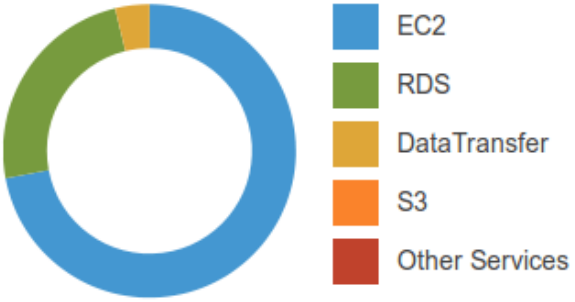
Stroškovni model

V tem poglavju najprej opišemo način zaračunavanja za uporabo storitev na obeh oblakih, nato opišemo in definiramo stroškovni model ter na isti graf vizualiziramo stroškovni model za oba oblaka in ga analiziramo.

V stroškovni model vključimo samo ceno podatkovne baze in ceno virtualnih strojev za izvajanje spletne trgovine, ker se je izkazalo, da omenjeni storitvi predstavljata največji delež stroškov. Ostale stroške za prenos podatkov, uporabo storitve za shranjevanje podatkov in uporabo storitve avtomatskega skaliranja ne vključimo v stroškovni model, ker je njihov delež pri končni ceni zanemarljivo majhen. Slika 9.1 prikazuje delež stroškov za storitve, ki smo jih uporabljali na oblaku AWS. Enako velja tudi za oblak GCP, saj tudi tu največji delež stroškov predstavljata podatkovna baza in virtualni stroji za izvajanje spletne aplikacije. Stroški vzpostavitve infrastrukture za testiranje v stroškovnem modelu niso všteti v končno ceno.

9.1 Način zaračunavanja na oblaku AWS

Oblak AWS za storitvi EC2 in RDS omogoča dva modela plačevanja: **na zahtevo** in **plačevanje za rezervirane virtualne stroje**. Pri plačevanju **na zahtevo** plačamo za vsako uro, ko se virtualni stroj izvaja, ne glede na to ali se virtualni stroji izvaja 1 minuto ali 60 minut. Model plačevanja za



Slika 9.1: Delež stroškov na oblaku AWS.

rezervirane virtualne stroje je enak modelu plačevanja **na zahtevo**, le da pri tem modelu plačamo za 1 ali 3 leta vnaprej in smo zaradi tega deležni popustov na ceno, kot je določena pri modelu plačevanja **na zahtevo** [30].

Pri izdelavi stroškovnega modela uporabljamo model plačevanja **na zahtevo**. Tabela 9.1 prikazuje cene za eno uro uporabe za tip instance virtualnega stroja **m3.medium** in **m3.large** za izvajanje spletne aplikacije, ki smo ju uporabili pri izvajanju meritev. Tabela 9.2 pa prikazuje cene za uro uporabe podatkovne baze za tipe instanc **db.m3.medium**, **db.m3.large** in **db.m3.xlarge**, ki smo jih uporabili za izvajanje meritev iz tabele 8.5 in 8.6.

9.2 Način zaračunavanja na oblaku GCP

Oblak GCP za storitev Compute Engine zaračuna za vsako minuto, ko se virtualni stroj izvaja. Minimalno zaračunajo za 10 minut uporabe. To pomeni, da če virtualni stroj izvajamo 1 ali 9 minut, nam storitev Compute Engine

Tip instance	Cena na uro v $
m3.medium	0.067
m3.large	0.133

Tabela 9.1: Cena za eno urno uporabo tipov instanc **m3.medium** in **m3.large** za storitev EC2.

Tip instance	Cena na uro v $
db.m3.medium	0.090
db.m3.large	0.185
db.m3.xlarge	0.370

Tabela 9.2: Cena za tipe instanc storitve RDS, ki smo jih uporabili za izvajanje meritev.

zaračuna za 10 minut uporabe. Po 10 minutah zaračuna za vsako minuto uporabe. Na spletni strani so cene navedene za eno uro uporabe in si moramo nato sami preračunati, koliko $ znese na minuto. Za uporabo storitve Compute Engine imamo na voljo tri cene: cena na uro za trajno uporabo, cena na uro za normalno uporabo, cena na uro brez trajne uporabe.

Cena na uro za trajno uporabo pomeni, da so v ceno vključeni popusti, če se virtualni stroj izvaja 100 % na mesec.

Cena na uro za normalno uporabo je cena za uro uporabe virtualne stroja, če se ta izvaja pod povprečno uporabo. Povprečna uporaba se določi čez vse uporabnike storitve Compute Engine.

Cena na uro brez trajne uporabe je cena, ki jo plačamo za virtualne stroje, ki se izvajajo manj kot 25 % na mesec.

Tabela 9.3 prikazuje **cene na uro brez trajne uporabe** za tipe instanc `n1-standard-1` in `n1-standard-2` za izvajanje spletne aplikacije, ki smo jih uporabili za izvajanje meritev iz tabel 8.3 in 8.4.

Tip instance	Cena na uro v $
n1-standard-1	0.050
n1-standard-2	0.100

Tabela 9.3: Cena za eno urno uporabo Compute Engine instanc.

Tip instance	Cena na uro v $
D8	0.58
D16	1.16
D32	2.31

Tabela 9.4: Cena za eno urno uporabo storitve Cloud SQL.

Oblak GCP za storitev Cloud SQL ponuja dva modela plačevanja: **po porabi** in **paketno plačevanje**. Pri **paketnem plačevanju** plačamo za vsak dan, ko se je podatkovna baza izvajala. **Paketno plačevanje** je primerno, če uporabljamo podatkovno bazo več kot 450 ur vsak mesec. Plačevanje **po porabi** pa je primerno za časovno krajša izvajanja, kjer plačamo za število ur ko se je podatkovna baza izvajala. Za izdelavo stroškovnega modela uporabimo model plačevanja **po porabi**.

Tabela 9.4 prikazuje cene za eno uro uporabe storitve Cloud SQL za tipe instanc, ki smo jih uporabili pri izvajanju meritev iz tabele 8.3 in 8.4.

9.3 Definicija stroškovnega modela

Stroškovni model analizira različne komponente stroškov infrastrukture in predstavlja analizo cenovnih prednosti glede na razpoložljive možnosti oblaka. Stroškovni model upošteva osnovne komponente, ki so vključene pri vzpostavljanju in upravljanju informacijske infrastrukture [31].

V magistrski nalogi stroškovni model predstavimo z grafom, ki prikazuje koliko je potrebno plačati za posamezno konfiguracijo sistema Q in kakšno kapaciteto lahko podpremo za ta denar. Matematično lahko stroškovni model predstavimo s funkcijo:

$$z(c(p, Q, d)) = k_{p,Q} \tag{9.1}$$

kjer je:

$c(p, Q, d)$ - funkcija, ki vrne ceno za konfiguracijo sistema Q za oblak p, z dolžino scenarija d,

$k_{p,Q}$ - kapaciteta za konfiguracijo sistema Q za oblak p, določena z, enačbo 6.1.

Zaradi različnega načina zaračunavanja za uporabo storitev posameznega oblaka, funkcijo $c(p, Q, d)$ iz enačbe 9.1 definiramo za vsak oblak posebej. Definicija za oblak GCP upošteva eno minutni interval zaračunavanja (enačba 9.2), definicija za oblak AWS pa eno urni interval zaračunavanja (enačba 9.3).

$$c(p, Q, d) = r_i(p_Q) \times p_{Q,n} + r_i(f_q) \times f_{Q,n} * \frac{d}{60} \tag{9.2}$$

$$c(p, Q, d) = r_i(p_Q) \times p_{Q,n} + r_i(f_q) \times f_{Q,n} \tag{9.3}$$

kjer je:

p_Q - tip instance za podatkovno bazo pri konfiguraciji sistema Q,

$p_{Q,n}$ - število virtualnih strojev za podatkovno bazo pri konfiguraciji Q,

d - dolžina scenarija,

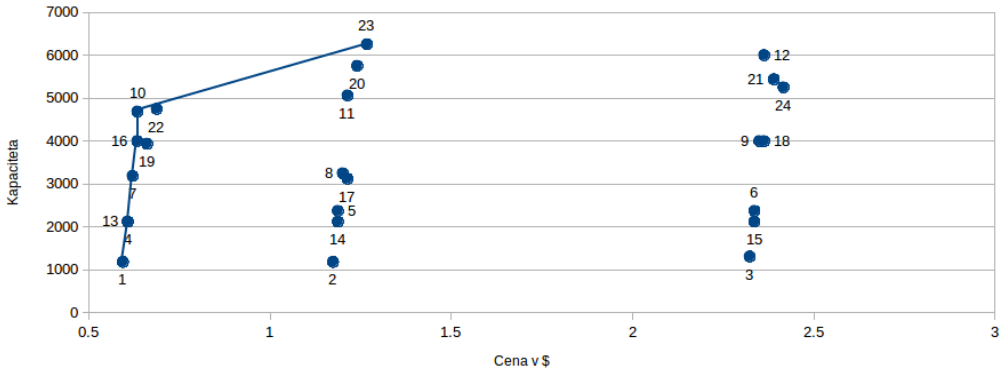
f_q - tip instance za izvajanje spletne aplikacije,

$f_{Q,n}$ - število virtualnih strojev za izvajanje spletne aplikacije,

$r_i(p_Q)$ - vrne ceno za eno urno uporabo za tip instance p_Q.

9.4 Vizualizacija stroškovnega modela

Za vizualizacijo stroškovnega modela smo najprej z uporabo enačbe $c(p, Q, d)$, tabel 9.1, 9.4, 9.3 in 9.2 za vsako meritev izračunali končno ceno, ki smo jo morali plačati za izvedbo ene meritve. Končna cena skupaj s kapaciteto predstavlja stroškovni model. Stroškovni model vizualiziramo za ročno in avtomatsko skaliranje.



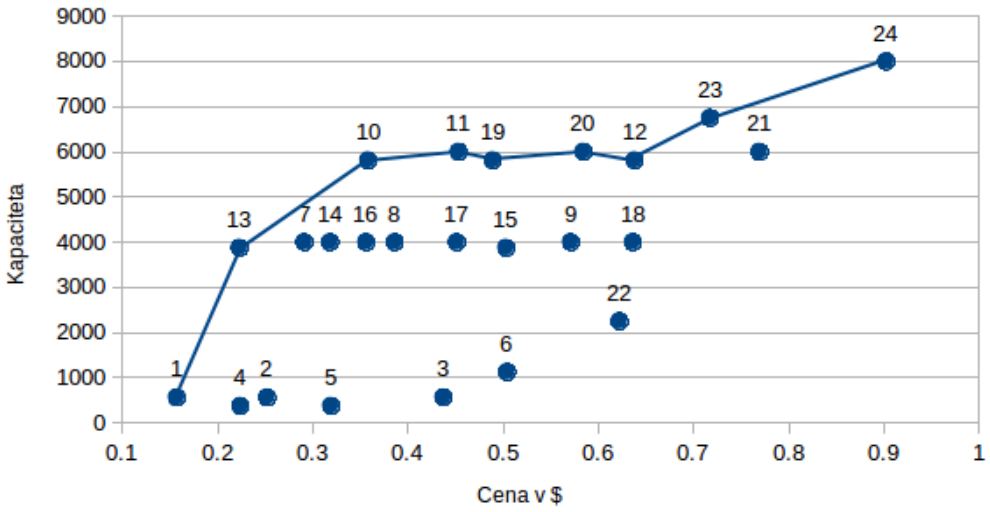
Slika 9.2: Vizualizacija stroškovnega modela za oblak GCP za ročno skaliranje. Črta prikazuje najbolj ugodne meritve, ki za najmanjšo ceno ponujajo največjo kapaciteto.

9.4.1 Vizualizacija za ročno skaliranje

Slika 9.2 prikazuje vizualizacijo stroškovnega modela za oblak GCP za ročno skaliranje, kjer ena točka ustreza eni meritvi iz tabele 8.3. Črta, ki povezuje točke, označuje meritve, s katerimi za najmanjšo ceno dosežemo največjo kapaciteto. Primer: če želimo podpreti 1000 virtualnih uporabnikov v 16-ih minutah z linearno naraščajočo obremenitvijo, raje izberemo konfiguracijo sistema, kot jo določa meritev 1, namesto meritve 2 ali 3, ker sta obe dražji od meritve 1.

Podobno vizualizacijo stroškovnega modela naredimo tudi za meritve z ročnim skaliranjem na oblaku AWS, ki jo prikazuje slika 9.3. Tudi pri tem grafu črta označuje meritve, s katerimi za najmanjšo ceno dosežemo največjo kapaciteto. Tudi na grafu 9.3 opazimo, da namesto konfiguracije sistema, kot jo določa meritev 4, raje uporabimo meritev 13, ker za isto ceno ponuja večjo kapaciteto.

Primerjava stroškovnega modela za oba oblaka za meritve z ročnim skaliranjem z grafom **??** pokaže, da je v večini primerov za namestitev spletne aplikacije bolje izbrati storitve oblaka AWS kot storitve oblaka GCP.

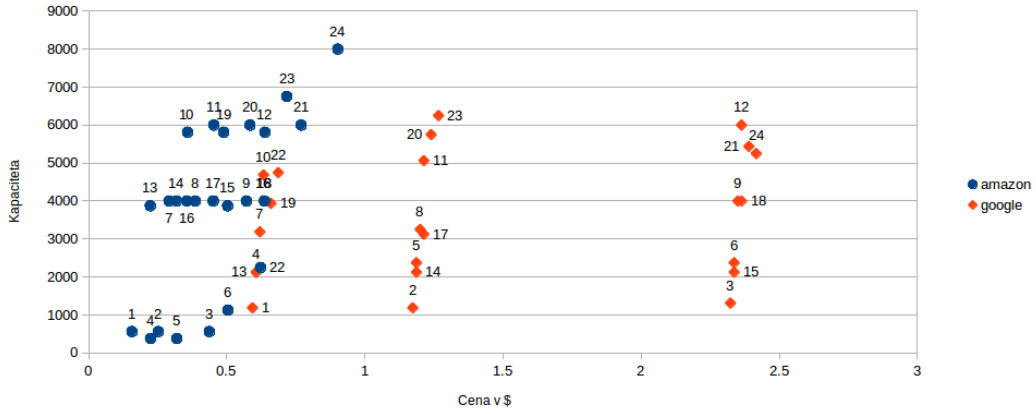


Slika 9.3: Vizualizacija stroškovnega modela za oblak AWS za ročno skaliranje. Črta prikazuje najbolj ugodne meritve, ki za najmanjšo ceno ponujajo največjo kapaciteto.

9.4.2 Vizualizacija za avtomatsko skaliranje

Slika 9.5 prikazuje vizualizacijo stroškovnega modela za meritve na oblaku GCP z vklopljenim avtomatskim skaliranjem. Črta na grafu označuje meritve, ki ponujajo največjo kapaciteto za najnižjo ceno. Primer: če želimo podpreti čimvišjo kapaciteto, bomo izbrali namestitveno konfiguracijo sistema, kot jo določa meritev 20, saj z njo dosežemo največjo kapaciteto. Vredno je tudi razmisliti o izbiri konfiguracije sistema, kot jo določa meritev 13, s katero sicer dosežemo malenkost manjšo kapaciteto, vendar je tudi precej ugodnejša.

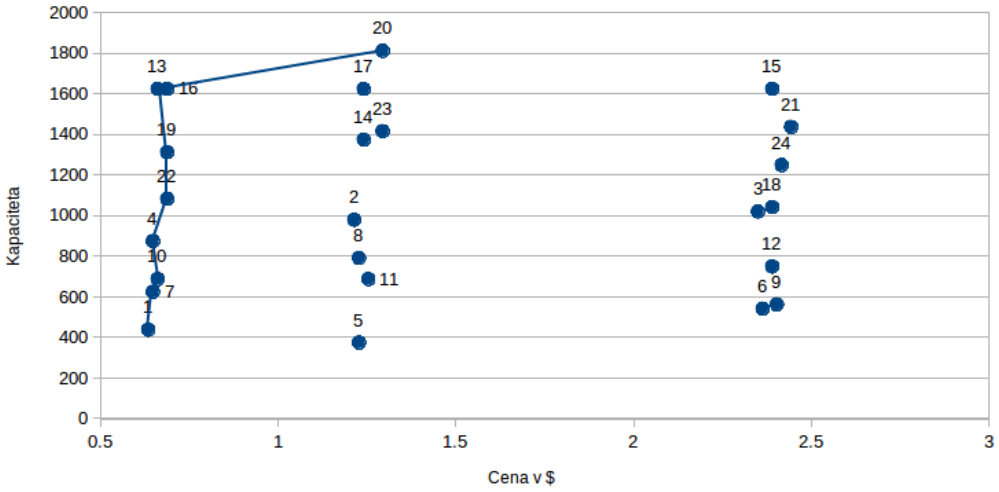
Slika 9.6 prikazuje vizualizacijo stroškovnega modela za meritve na oblaku AWS z vklopljenim avtomatskim skaliranjem. Tudi na tem grafu črta označuje meritve, ki ponujajo navečjo kapaciteto za najnižjo ceno. Opazimo, da ima veliko meritev enako ceno, npr. meritve 4, 7 in 10. V tem primeru bi za 16 minutno obremenitev z linearnim povečevanjem izbrali namestitveno konfi-



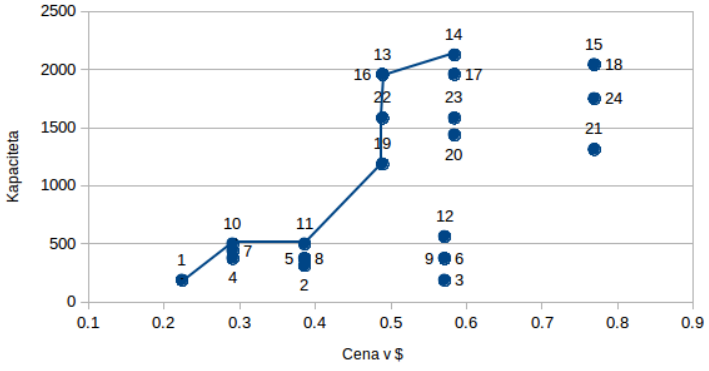
Slika 9.4: Vizualizacija stroškovnega modela za oba oblaka za meritve z ročnim skaliranjem, na isti graf.

guracijo sistema, kot jo določa meritev 10.

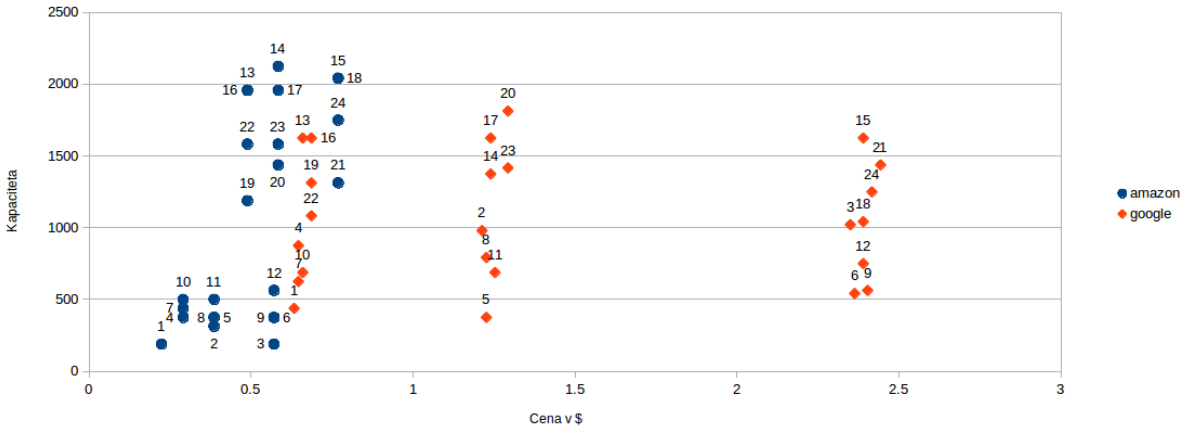
Na sliki 9.7 vizualiziramo stroškovni model za avtomatsko skaliranje za oba oblaka na isti graf. Podobno kot pri vizualizaciji meritev z ročnim skaliranjem na isti graf, tudi pri vizualizaciji meritev z avtomatskim skaliranjem na isti graf opazimo, da na oblaku AWS dosežemo večje kapacitete za nižjo ceno, kot na oblaku GCP.



Slika 9.5: Vizualizacija stroškovnega modela za meritve na oblaku GCP z vklopljenim avtomatskim skaliranjem. Črta označuje meritve, ki ponujajo največjo kapaciteto za najnižjo ceno.



Slika 9.6: Vizualizacija stroškovnega modela za meritve na oblaku AWS z vklopljenim avtomatskim skaliranjem. Črta označuje meritve, ki ponujajo največjo kapaciteto za najnižjo ceno.



Slika 9.7: Vizualizacija stroškovnega modela na isti graf za meritve na obeh oblakih z vklopljenim avtomatskim skaliranjem.

Poglavje 10

Sklepne ugotovitve

V magistrski nalogi smo raziskovali novo, še ne raziskano področje testiranja skalabilnosti in analize stroškov povezanih s povečano obremenitvijo. Z razvojem orodij, vzorčne aplikacije, definicije metrik in stroškovnega modela smo postavili temelje, ki jih lahko uporabimo tudi za testiranje drugačnega tipa aplikacij, kot je naša vzorčna aplikacija. Razvita orodja smo uporabili za testiranje različnih konfiguracij sistema, rezultate pa ovrednotili z definiranimi metrikami in stroškovnim modelom. Pri izvajanju meritev smo naleteli na veliko vprašanj, ki bi jih lahko še raziskali, npr.:

- testiranje namestitvenega modela nadrejeni-podrejeni za podatkovno bazo,
- testiranje daljših scenarijev od 16 minut za meritve z vklopljenim avtomatskim skaliranjem,
- testiranje ostalih tipov instanc za podatkovno bazo in tipov instanc za izvajanje spletne aplikacije,
- testiranje meritev z podatkovno bazo tipa NoSQL.

Pri izvajanju meritev smo naleteli tudi na nekaj ozkih grl z omrežjem in programsko opremo, ki jo uporabljamo za izvajanje spletne aplikacije. Problem ozkega grla z omrežjem smo rešili tako, da smo izvajanje programa

za obremenitveno testiranje iz našega osebnega računalnika preselili v oblak, na katerem smo izvajali meritve. Težave s programsko opremo, ki se pokažejo šele ob povečani obremenitvi, pa smo rešili z optimizacijo nastavitev.

Pri izvajanju meritev z vklopljenim avtomatskim skaliranjem nismo bili prepričani, na kakšno vrednost naj nastavimo velikost bazena povezav do podatkovne baze, za katero smo ugotovili, da najbolj vpliva na končno kapaciteto sistema. Analiza rezultatov meritev z vklopljenim avtomatskim skaliranjem je pokazala, da je najbolje omenjeno velikost nastaviti kar na vrednost omejitve podatkovnih baz maksimalnega števila sočasnih povezav.

Analiza stroškovnih modelov za meritve za oba oblaka pokaže, da smo na oblaku AWS v splošnem dobili višje kapacitete s cenejšimi konfiguracijami sistema. Primerjava cen tipov instanc za izvajanje spletne aplikacije sprva da občutek, da je oblak AWS dražji, vendar temu ni tako. Primerjava cen tipov instanc storitev za podatkovno bazo, ki smo jih uporabili za izvajanje meritev pokaže, da je razlika med cenami tako velika, da naredi meritve za oblak GCP veliko dražje od meritev na oblaku AWS. K dražji ceni meritev na oblaku GCP pripomore tudi večje število virtualnih strojev, ki jih storitev avtomatskega skaliranja zažene na oblaku GCP, da podpre določeno obremenitev.

Ob dejstvu, da je storitev avtomatskega skaliranja še vedno v beta razvojni fazi in zanj dogovor o ravni storitve (angl. *Service Level Agreement - SLA*) ne velja, iz analize stroškovih modelov opazimo, da je kapaciteta v splošnem na oblaku GCP manjša kot na oblaku AWS. Slednja ugotovitev nakazuje na to, da je zmogljivost in delovanje storitve avtomatskega skaliranja pri oblaku GCP manj stabilno kot pri oblaku AWS.

Literatura

[1] Googlov TOC kalkulator, Dostopno na: https://cloud.google.com/products/calculator/.

[2] X. Xu, H. Jin, S. Wu, L. Tang, Y. Wang, Urmg: Enhanced cbmg-based method for automatically testing web applications in the cloud, Tsinghua Science and Technology 19 (1) (2014) str. 65–75.

[3] J. Gao, X. Bai, W.-T. Tsai, Cloud testing-issues, challenges, needs and practice, Software Engineering: An International Journal (SEIJ), Delhi Technological University 1 (1) (2011) str. 9–23.

[4] PushToTest, Dostopno na: http://www.pushtotest.com.

[5] CloudTesting, Dostopno na: http://www.cloudtesting.com/.

[6] SOASTA, Dostopno na: http://www.soasta.com/.

[7] iTKO, Dostopno na: http://www.itko.com/.

[8] W. Zhang, S. Wang, W. Wang, H. Zhong, Bench4q: A qos-oriented e-commerce benchmark, in: Computer Software and Applications Conference (COMPSAC), 2011 IEEE 35th Annual, 2011, pp. str. 38–47.

[9] T. P. P. C. (TPC), Tpc Benchmark W, Dostopno na: http://www.tpc.org/tpcw/spec/tpcw_v16.pdf.

[10] M. Becker, S. Lehrig, S. Becker, Systematically deriving quality metrics for cloud computing systems, in: Proceedings of the 6th ACM/SPEC

International Conference on Performance Engineering, ICPE '15, ACM, New York, NY, USA, 2015, pp. str. 169–174.

[11] Andre B. Bondi. Foundations of Software and System Performance Engineering. Addison Wesley, 2015.

[12] J. Murty, Programming Amazon Web Services: S3, EC2, SQS, FPS and Simple DB (2008).

[13] AWS Instance types, Dostopno na: http://aws.amazon.com/ec2/instance-types/.

[14] Relational Database Service, Dostopno na: https://aws.amazon.com/rds/details/.

[15] RDS dokumentacija, Dostopno na:https://aws.amazon.com/rds/mysql/details/.

[16] S3, Dostopno na: https://aws.amazon.com/s3/details/.

[17] Google Cloud Platform, Dostopno na: https://cloud.google.com/.

[18] Google Cloud Documentation, Dostopno na: https://cloud.google.com/compute/docs.

[19] HP LoadRunner software, Dostopno na: http://www8.hp.com/h20195/V2/GetPDF.aspx/4AA1-2118ENW.pdf.

[20] E. H. Halili, Apache JMeter : A Practical Beginner's Guide to Automated Testing and Performance Measurement for Your Websites., Packt Publishing, 2008.

[21] NIST Cloud Metrics, Dostopno na: http://www.nist.gov/itl/cloud/upload/RATAX-CloudServiceMetricsDescription-DRAFT-20141111.pdf.

[22] NIST, The NIST Definition of Cloud Computing, Dostopno na: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf.

[23] N. R. Herbst, S. Kounev, R. Reussner, Elasticity in cloud computing: What it is, and what it is not, in: Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13), USENIX, San Jose, CA, 2013, pp. str. 23–27.

[24] S. Lehrig, H. Eikerling, S. Becker, Scalability, elasticity, and efficiency in cloud computing: A systematic literature review of definitions and metrics, in: Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures, QoSA '15, ACM, New York, NY, USA, 2015, pp. str. 83–92.

[25] X. Zheng, P. Martin, K. Brohman, L. D. Xu, Cloudqual: A quality model for cloud services, Industrial Informatics, IEEE Transactions on 10 (2) (2014) str. 1527–1536.

[26] TPC-W implementacija, Dostopno na: `http://pharm.ece.wisc.edu/tpcw.shtml`.

[27] c3p0, Dostopno na: `http://www.mchange.com/projects/c3p0/`.

[28] Vmesnik za plačevanje, Dostopno na: `https://arcane-meadow-6418.herokuapp.com/`.

[29] GitHub repozitorij z izvorno kodo razvitih orodij in aplikacij, Dostopno na: `https://github.com/ivansek-magistrska`.

[30] Dokumentacija za AWS oblak, Dostopno na: `https://aws.amazon.com/documentation/`.

[31] D. Ajeh, J. Ellman, S. Keogh, A cost modelling system for cloud computing, in: Computational Science and Its Applications (ICCSA), 2014 14th International Conference on, 2014.